2011-09

Residual network data structures in Android devices

Cardwell, Gregory S.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

RESIDUAL NETWORK DATA STRUCTURES IN ANDROID DEVICES

by

Gregory S. Cardwell

September 2011

Thesis Co-Advisors: Robert Beverly
Simson Garfinkel

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23-09-2011	Master's Thesis	2009-07-01—2011-09-23

4. TITLE AND SUBTITLE

Residual Network Data Structures in Android Devices

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Gregory S. Cardwell

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. **IRB Protocol Number: XXXX**

14. ABSTRACT

The emergence and recent ubiquity of Smartphones present new opportunities and challenges to forensic examiners. Smartphones enable new mobile application and use paradigms by being constantly attached to the Internet via one of several physical communication media, e.g. cellular radio, WiFi, or Bluetooth. The Smartphone's storage medium represents a potential source of current and historical network metadata and records of prior data transfers. By using known ground truth data exchanges in a controlled experimental environment, this thesis identifies network metadata stored by the Android operating system that can be readily retrieved from the device's internal non-volatile storage. The identified network metadata can ascertain the identity of prior network access points to which the device associated. An important by-product of this research is a well-labeled Android Smartphone image corpus, allowing the mobile forensic community to perform repeatable, scientific experiments, and to test mobile forensic tools.

15. SUBJECT TERMS

Android Operating System, Forensics, Smartphones, Network Data Structures, Network Forensics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	117	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

RESIDUAL NETWORK DATA STRUCTURES IN ANDROID DEVICES

Gregory S. Cardwell
Lieutenant Commander, United States Navy
B.S., College of the Ozarks, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2011

Author: Gregory S. Cardwell

Approved by: Robert Beverly
Thesis Co-Advisor

Simson Garfinkel
Thesis Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The emergence and recent ubiquity of Smartphones present new opportunities and challenges to forensic examiners. Smartphones enable new mobile application and use paradigms by being constantly attached to the Internet via one of several physical communication media, e.g. cellular radio, WiFi, or Bluetooth. The Smartphone's storage medium represents a potential source of current and historical network metadata and records of prior data transfers. By using known ground truth data exchanges in a controlled experimental environment, this thesis identifies network metadata stored by the Android operating system that can be readily retrieved from the device's internal non-volatile storage. The identified network metadata can ascertain the identity of prior network access points to which the device associated. An important by-product of this research is a well-labeled Android Smartphone image corpus, allowing the mobile forensic community to perform repeatable, scientific experiments, and to test mobile forensic tools.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms

AP Access Point

ASCII American Standard Code for Information Interchange

ADB Android Debug Bridge

AT Attention

BGA Ball Grid Array

BS Base Station

BSC Base Station Controller

CID Cell Identification

DHCP Dynamic Host Configuration Protocol

EEPROM Electrically Erasable Programmable Read-Only Memory

EXT Extended File System

FFL Flash File System

FTL Flash Transition Layer

GID Group Identification

GSM Global System for Mobile

IEEE Institute of Electrical and Electronics Engineers

IMEI International Mobile Equipment Identity

IMSI International Mobile Subscriber Identity

IP Internet Protocol

LAC Local Area Code

LAN Local Area Network

JTAG Joint Test Action Group

MAC Media Access Control

MTD Memory Technology Device

MCC Mobile Country Code

MNC Mobile Network Code

MSC Mobile Switching Center

MCC Mobile Country Code

MNC Mobile Network Code

MLC Multi-Level Cell

NAT Network Address Translation

NIST National Institute of Standards and Technology

NPS Naval Postgraduate School

OS Operating System

RF Radio Frequency

RAM Random-Access Memory

ROM Read-Only Memory

SLC Single-Level Cell

SD Secure Digital

SIM Subscriber Identity Module

SDK Software Development Kit

SA Spectrum Analyzer

SSID Service Set Identifier

TAP Test Access Port

TCK Test Clock Input

TDI Test Data Input

TDO Test Data Output

TMS Test Mode Select

TRST Test-reset Input

TDD Time Division Duplex

TSOP Thin Small-Outline Packages

TCP Transport Control Protocol

UID User Identification

USB Universal Serial Bus

URL Uniform Resource Locator

WPAN Wireless Personal Area Network

YAFFS Yet Another Flash File System

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

This thesis would not have been possible without the guidance and instruction from Dr. Rob Beverly and Dr. Simson Garfinkel. I would especially like to recognize Dr. Beverly for his tremendous enthusiasm, unparalleled patience and constant understanding.

Most of all, I would like to thank my wife Nichole, who spent many nights and weekends alone raising our two wonderful children, Nicholas and Brianna. Without her support and dedication, this effort would not have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Smartphones have seen unprecedented growth and are not showing any signs of stagnating. The International Data Corporation expects 450 million Smartphones shipped in 2011 compared to the 303 million shipped in 2010 [1]. Smartphones enable constant internet attachment via one of several physical communication media. Through the existing cellular network, these devices provide consumers data services. They also provide 802.11 (§2.3.2) wireless interfaces to send data over local area networks and 802.15 (§2.3.3) wireless personal access network interfaces to transfer data to other Bluetooth-enabled devices. With the increasing number of mobile applications available, Smartphones are now powerful personal digital assistants with telephony and data capabilities. Consumers use these devices to take photographs, send important emails, interact with social networks, surf the internet, etc.

Smartphones are used alike by law-abiding citizens and criminals. If a Smartphone belonging to a criminal or terrorist is captured, there are numerous commercial and open source tools available to extract personal information: call logs, emails, and other kinds of information (§2). In addition to contact information and photos, authorities increasingly want to know where the phone was physically operating and where it gained access to networks.

Mobile forensic tools and efforts have mostly concentrated on personal information data that resides on the Smartphone. This thesis will attempt to extract network metadata that will provide investigators previous knowledge of a suspects historical network access points.

1.1 Mobile Device's Historical Network Access Points

This thesis will focus on residual network metadata from Android non-volatile memory. Although, dumping the volatile memory of the captured mobile device may provide authorities valuable data concerning network access points. These data are dynamically changing and may provide information on the last beacon probe the device received from a wireless network access point used: it may also provide cellular base station data from the current physical area's cellular tower. Additional information may be found that reveals the suspect's previous whereabouts.

The non-volatile memory characteristics of Smartphones provides significant valuable historical data. NAND flash is currently the de facto standard for Smartphone non-volatile storage. Flash

has a limited lifespan that requires the file system to carefully perform read and write operations to the physical storage. This wear leveling technique allows data that has been marked for deletion to persist on the storage medium potentially making substantial information available for analysis. This opportunity to extract residual network data from non-volatile memory is a fundamental change in traditional forensics where network data is usually stored in volatile memory.

There exist multiple forms of data that can be useful to forensic examiners. Cellular base stations transfer their metadata to Smartphones for access purposes. This data contains information to the particular cellular tower that can be physically geolocated. Wireless routers, commonly found in businesses and homes share their unique Media Access Control (MAC) address and Service Set Identifier (SSID) with the mobile devices. This information, used in conjunction with open source wireless geographic engines can identify a known wireless router's location, which can be used to determine the Smartphone's previous physical locations [2].

Other network data can also help with forensic investigations. Bluetooth MAC addresses of associated devices will persist on memory. Subscriber Identity Modules (SIM) cards required in GSM cellular networks will also save historical data to memory. This data is useful if a criminal uses the mobile device with multiple SIM cards to attempt to hide his identity.

Binary Internet Protocol (IP) addresses also persist on the device's non-volatile memory. This data can prove to be useful if the device communicates with a wireless router that provided addresses not found in the RFC 1918 address allocation range [3].

1.2 Research Questions

This thesis asks one primary question: do sufficient residual artifacts exist on mobile devices to extract enough data to identify the device's previous network access points? The primary objective can be achieved through answering the secondary questions:

- Does a mobile device store network-relevant information in non-volatile storage?
- What encoding techniques does the mobile device use to store the information?
- How does flash wear-leveling techniques affect forensic data collection?
- Do the data reveal network usage patterns?

To answer these questions, a carefully controlled testing environment was used to transfer data between Android Smartphones and multiple network access point (cellular, wireless, and Bluetooth). This experiment provided a ground truth data set to determine any identifiers that can be found to commonly extract network relevant information from devices with unknown provenance.

1.3 Significant Findings

Using known ground truth data, two Android Smartphones were used in a controlled environment, imaged and analyzed for the known ground truth network metadata. A total of seven controlled experiments were performed producing 7 Nexus One images and 7 Nexus S images. The images contain YAFFS2 partitions and the Nexus S contain three EXT4 partitions. Analysis of these images reveals:

- `userdata` partitions contain allocated and non-allocated pages with wireless router Service Set Identifiers (SSID), which are ASCII coded names provides to wireless access points.

- `userdata` partitions contain allocated and non-allocated pages with wireless router Subscriber Identity Modules (SIM).

- `userdata` partitions contain allocated and non-allocated pages with DHCP ACKs from wireless routers. DHCP ACKs contain the unsigned 32-bit IP address of the Android Device and the wireless router providing network services.

- `userdata` partitions contain allocated and potentially non-allocated base station metadata, including the Mobile Network Code (MNC), Mobile Country Code (MCC), Local Area Code (LAC) and Cell Identification (CID). These four base station metadata can be used to physically locate a cellular tower.

- `userdata` partitions contain allocated wireless router Media Access Control (MAC) addresses coded in ASCII.

- `userdata` partitions contain allocated and non-allocated pages with Bluetooth MAC addresses of devices paired with the phone.

1.4 Thesis Structure

This remainder thesis is organized as follows:

- Chapter 2 discusses prior work in mobile forensics and discusses network protocols and the Android operating system and its components.
- Chapter 3 describes the hardware and its configuration used in the experiments and delineates each produced image.
- Chapter 4 provides the results of the Android image analysis.
- Chapter 5 contains conclusions drawn from Chapter 4 and recommended future work.

CHAPTER 2:
Background and Related Work

2.1 Mobile Forensics

The last decade has seen unprecedented growth in the mobile computing market. Sophisticated communications capabilities have been incorporated into new form factors such as portable music players, tablets and Smartphones. These devices provide consumers the functionality of personal computers and freedom of mobility, yet cost significantly less than traditional personal computers.

Smartphones are of particular interest in forensics due to their various physical communication media (e.g., GSM radio, WiFi, Bluetooth, etc), variety of communication applications (e.g., electronic mail, messaging, social network applications), and their abundance of portable fixed storage. Mobile devices have become ubiquitous tools that are carried by owners on a daily basis. In 2010, over 271 million Smartphones were sold worldwide [4]. Unfortunately, the rich functionality and ease of mobility that is so attractive to everyday consumers is also proving beneficial to criminals. Smartphones are used in the commission of many crimes, including theft, child exploitation and extortion. Malicious applications, commonly referred to as malware, are finding a niche in the mobile market. Malware is able to download and forward data from a mobile device to remote servers, place clandestine phone calls and text messages to numbers that charge rates to the compromised Smartphone's bill [5]. For investigators, the only consolidation is that the ability to retain malicious data or a wide array of personal data, mobile devices can represent a rich source of evidence for forensic examiners.

The National Institute of Standards and Technology (NIST) defines digital forensics as "the application of science to the identification, collection, examination, and analysis, of data while preserving the integrity of the information and maintaining a strict chain of custody for the data" [6]. The forensic science applied to mobile devices is still in its infancy. Device capabilities are increasing every year, and forensic techniques struggle to keep apace. For example, the introduction of Windows pocket PCs and Apple's iPhone have blurred any functionality difference between mobile devices and personal computers — requiring new tools and techniques for forensic examination. With a new generation of Smartphones populating the market every year, device storage capacity, functionality, and architecture is rapidly evolving, creating issues for

the forensic community.

Smartphones use a variety operating systems (OS's) and employ many different hardware interfaces (e.g., USB, proprietary connectors, etc). Across the previously mentioned 271 million Smartphones sold in 2010, four different operating systems were predominant: Apple's iOS, Google's Android, RIM's Blackberry and Nokia's Symbian. Each Smartphone presents unique characteristics for forensic examiners. In addition to the heterogeneity of operating systems, some devices use proprietary USB connectors, requiring examiners to have a different cable for each specific device in order to acquire pertinent forensic data. The examiner's forensic software must also be updated often to maintain compatibility with updated OS versions and for every new Smartphone device introduced – a daunting and expensive task.

Further complicating the forensic process, access to a mobile device's memory can be restricted by the operating system. For instance, Android uses the standard UNIX User (UID) and Group (GID) file permissions for each application and, by default, restricts root access.

NIST SP800-101 contains guidelines for conducting forensics on cellular devices [7]. Cellular devices present unique challenges and certain aspects of these devices must be considered to maintain data integrity on the device. The NIST document outlines key topics for cell phone forensic examiners, with three topics unique to mobile devices: Forensic Tools, Preservation and Acquisition. Cellular devices must be handled differently than standard computing devices because incoming data or phone calls may contaminate the data residing on the device after recovery or even during the analysis. As previously mentioned, there are multiple form factors and interfaces which create challenges when preparing mobile forensic tools and acquiring data from the devices.

This chapter discusses several methods to overcome these obstacles and extract data from the mobile device's non-volatile storage via physical or logical acquisition to support forensic examination. Some of these methods, including the Nandroid logical acquisition technique used in this study, violate the forensic principles outlined in NIST SP 800-101. Methods to acquire data from an evidence source should make every attempt to remove the data without modifying the state of the mobile device. The methods used in this thesis do modify the mobile devices's state, but it is currently the only known acquisition technique currently available that allows us to acquire low level network data structures and unallocated data. The remainder of this chapter provides an overview of the Android OS, the physical and logical acquisition techniques are discussed.

2.2 Android Overview

The Android Operating System is based on the open source Linux 2.6 kernel. The Android kernel is similar to normal Linux kernels with the exception that a flash memory driver is provided to support internal memory (§2.2.1).

The Linux kernel is designed to support standard block storage devices. Android devices use flash for non-volatile storage, so they must use an layer of abstraction to facilitate use of the Linux kernel. This layer is referred to as the Memory Technology Device (MTD) system. The MTD provides block interface, ECC, wear-leveling and other functions required to support flash memory [8].

The Google Nexus One Smartphone MTD partitions are shown in Table 2.1. These partitions are only representative of the HTC Nexus One as partitions vary and are manufacturer specific, but the Nexus One partitions show how the MTD presents the flash partitions to the kernel. The `size` column stipulates the actual size of the partition. The `erasesize` column identifies the the flash block size in decimal [8]. NAND flash can only perform erase operations at the block level, §2.2.1. The `userdata and system` partitions are the main subject of this thesis as they store data from user interaction with the device, and interaction with external network nodes. However, all partitions should be accounted for as they are easily manipulated through various tools which will be shown in the logical acquisition method used in this thesis.

partition	size	erasesize	name
mtd0	000e0000	00020000	“misc”
mtd1	00400000	00020000	“recovery”
mtd2	00380000	00020000	“boot”
mtd3	09100000	00020000	“system”
mtd4	05f00000	00020000	“cache”
mtd5	0c440000	00020000	“userdata”

Table 2.1: Nexus One MTD Partitions.

2.2.1 Flash Memory

Mobile devices use flash as secondary memory. Secondary memory is slower than other memory found in the device, but is usually the only form of storage that can retain data without a power source. Flash memory is a variation of Electrically Erasable Programmable Read-Only Memory (EEPROM). Like EEPROM, flash memory is non-volatile, meaning that it retains data

after it is unpowered. Flash differs from EEPROM in that it is byte-writable. Flash erase operations can only erase blocks of memory, however. Flash media allows two states: erased and non-erased. The erased state is all ones or all zeroes depending on the media. Data can only be written to bytes that are in an erased state.

Flash cells are floating-gate metal-oxide-semiconductor (MOS) transistors where the floating gate (FG) is insulated by dielectrics and governed by a control gate (CG). The dielectric between the gates is a triple-layer of oxide-nitride-oxide. The layer between the FG and the transistor channel is a thin layer of oxide. The dielectric layers provide the isolation to the FG to retain a charge, and are thin enough to be penetrated. A process called channel hot electron injection allows an electron to penetrate the thin oxide layer to give the FG a negative charge. A positively charged state indicates a logical 1 and the existence of stored electrons in the floating gate indicates a negatively charged state relating to a logical 0 [9].

Flash cells can be Single Level Cells (SLC) or Multi-Level Cells (MLC). SLC are older forms of flash that can store one bit per cell. MLC store more than one bit per cell and typically can represent four different states. MLC presents greater latency and a decreased lifetime from SLC [10]. The flash cells typically last on the order of 1×10^5 writing cycles [9]. The lifetime expectations present engineering challenges to the abstraction layer that interacts with the operating system. The physical properties of flash can be found in [10].

Mobile devices utilize NAND flash memory for secondary memory purposes. NAND flash provides increased storage capability and is relatively cheaper than other types of Flash. NAND cells are connected in a series, where read and write operations can only be performed on the entire series, not at the byte level. To erase a NAND cell, the entire series must be erased. The physical nature of NAND flash is extensively covered in [10].

Operating systems expect magnetic disk drives for secondary memory, so an abstract layer needs to be implemented between the physical flash memory and the software based operating system. The abstraction can be in the form of a flash transition layer (FTL), allowing the OS to use a traditional file system, or a flash file system (FFS). A FFS allows the OS to perform write and erase operations directly to flash memory without the need for a translation layer. The Android OS initially launched with the YAFFS2 file system (covered in the following section).

New versions of Android devices use both YAFFS and the Linux EXT4 file system. EXT4 requires the use of an FTL to perform read and write operations on the device's flash memory.

The FTL is a driver that presents flash memory as a block device to the OS. It accomplishes that abstraction by creating virtual blocks of data called sectors. When the OS tells the FTL that data needs to write to memory, the FTL provides the OS with a specific sector. The FTL keeps track of the sector given to the OS and physically writes data at a free location on flash memory. The FTL then maps the actual physical location of the data to the logical location the OS expects to see. Although Android's MTD has a software FTL, it is likely that EXT4 will be accessed through a hardware FTL for performance and intellectual property protection.

Due to the lifetime expectations of a flash cell, the FTL and FFS write to the flash media using a wear leveling. Wear leveling is a technique that writes rewritten logical blocks to separate physical blocks until all blocks have been utilized. In this way, no page or block degrades faster than the system as a whole.

This study makes both "physical" and "logical" memory dumps. A physical memory dump, or physical image, is a complete copy of all bytes retained in flash memory. The copy will retain all pages and the out-of-band data for each page as covered in the next section. A logical memory dump is a copy of the files the file system currently indicates as allocated. Intuitively, a logical dump will provide current files, while physical dumps will provide current files along with fragments of some previously used files.

2.2.2 Yet Another Flash File System

YAFFS was released in 2002. It is the main file system for devices using Android operating systems prior to 2011 and a secondary file system for newer devices [11]. YAFFS was designed for low-power portable devices and incorporates the following strategies for portability and simplicity:

- Single threaded model. YAFFS has on a per-partition lock, a design that is easy to implement and verify. The YAFFS Direct Interface uses a single lock for all partitions.
- Log structure makes for a simple allocation strategy and garbage collector.
- A modular, layered design, which is easier to understand and debug.

The smallest unit of memory in YAFFS is termed a `chunk`, which usually coincides to the size of the device's NAND flash page. A chunk can have two different chunk types, `data chunks` and `object headers`. Data chunks hold file contents, while the object header holds metadata regarding the object. Each `chunk` has the following associated metadata values:

- `ObjectId`: Identifies to which object the chunk belongs
- `ChunkId`: Identifies where in the file this chunk belongs.
- `Byte Count`: Number of bytes of data if this is a data chunk.
- `Sequence Number`: provides a way of organizing the log in chronological order [11].

YAFFS2 is a log structured file system. It does not write to markers that identify if a chunk is marked for deletion. Instead, it performs sequential writing to blocks. With use of the sequence number, it builds the file system state with backwards scanning. While scanning backwards, the most recently written chunk with a `ObjectId:ChunkId` pair will be considered current and the remaining matching pairs are considered deleted [11].

YAFFS2 determines its garbage collection method by the number of erased blocks available for use. If there are many erased blocks, YAFFS2 performs passive garbage collection, whereby it collects blocks with very few chunks in use. If there are few erased blocks, YAFFS2 performs an aggressive garbage collection where works harder to free more space. YAFFS2 block states are described in Table 2.2.

YAFFS2 performs garbage collection by scanning blocks. If YAFFS2 finds a block worth collecting, it creates a new block, copies the chunks in use, then erases the old block and marks it ready for use. YAFFS2 serially logs blocks from the erased blocks. By the time a block gets erased and then allocated again, a level of wear leveling occurs. Wear leveling spreads the writes and erases to each block, a vital requirement when using flash, as flash memory blocks have a finite number of writes and erases.

The YAFFS2 author describes the two heuristics for garbage collection as an attempt to delay garbage collection to reduce the amount of collection performed to increase average system performance [11]. This garbage collection scheme provides opportunity to retrieve remnant data not recognized by the file system.

2.2.3 EXT4 Filesystem

In December 2010, Google announced it would use the EXT4 file system in place of YAFFS2 in Android 2.3 and later versions [12]. EXT4 is the current Linux kernel file system. The move to EXT4 was prompted by the fact that YAFFS is single threaded and newer devices with multi-cores perform better with a multi-threaded file system [8].

State	Summary
UNKNOWN	The block state is not yet known.
NEEDS_SCANNING	During pre-scanning it has been determined that this block has something on it and needs scanning.
SCANNING	This block is currently being scanned.
EMPTY	This block has nothing in it (is erased) and may be used for allocation. The block can get to this state by being erased.
ALLOCATING	This is the block currently being used for allocation by the chunk allocator.
FULL	All the chunks in this block have been allocated and at least one chunk contains useful information that has not yet been deleted.
DIRTY	All the chunks in this block have been allocated and all have been deleted. The block might have been in the FULL or COLLECTING state before this. This block can now be erased and returned to EMPTY state.
CHECKPOINT	This block has checkpoint data in it. When the checkpoint is invalidated this block will be erased and returned to the EMPTY state.
COLLECTING	This block is being garbage collected. As soon as all live data has been inspected this block becomes DIRTY.
DEAD	This block has been retired or was marked bad. This is a terminal state.

Table 2.2: YAFFS2 Block States. From [11]

The EXT4 filesystem was introduced in 2007 and made numerous improvements over its predecessors. Most notable for this research, was the addition of extents and block pre-allocation. Extents are descriptors which represents a range of contiguous physical blocks. They provide an efficient data structure to map physical blocks to logical blocks.

As the name implies, EXT4's preallocation technique allows a file to have pre-allocated blocks, avoiding fragmentation if the file is subsequently expanded. EXT4 implements this technique by using the extent length field to indicate whether an extent contains initialized or uninitialized data. If the field indicates uninitialized data, then when a read occurs, the blocks are filled with zeroes. For further discussion on EXT4 improvements over EXT2 and EXT3 filesystems, see [13].

2.2.4 Prior Work

In his 2009 masters thesis, Regan observed the effects of write and read operations based on a YAFFS emulator [10]. He noted that the emulator, after completing a file delete operation, retained the intact file and added two new pages. The first new page contained object header updates and the second page was labeled unlinked and updated the MAC times, user id, group

id and set the remaining object metadata to zero. He also found that when a file was overwritten or partially overwritten, YAFFS2 retained the original file, wrote the overwritten file on a new page and updated two new pages similar to the delete operation. This behavior is consistent with the YAFFS2 wear leveling and garbage collection techniques, which provide unique data acquisition capabilities from a YAFFS file system. YAFFS2 allows deleted file retrieval and possibly multiple file retrievals based on the number of writes the file committed to memory [10].

In their 2010 work, Fairbanks Et al. noted curious behavior by the EXT4 file system. They note that EXT4 superblocks are different than previous EXT versions, which pose a problem with using existing forensic tools on EXT4. They show that the open source Sleuthkit took presents invalid data as Sleuthkit's algorithms were compiled for EXT2 superblocks. Another interesting outcome is that the group created an EXT4 filesystem and completely filled it with 4KB files containing the word "TEST". The files were deleted, then a large file filled with null values was written to the filesystem. They found 22MB of deleted data although the file system tells the user that the file is filled with zeroes [14].

Reardon Et al. introduced three mechanisms for secure deletion in their 2011 work. To test their solutions, they created an experiment that tested the "deletion latency" on Android devices using the YAFFS file system. They compiled a modified Linux kernel that logged data about block allocations and chunk writes. Using a HTC Nexus One Android device, they normally used the device for 670 hours. Defining block reallocation as the time a block is allocated until its reallocation by the garbage collector, their data showed that average block reallocation of about 44 hours with a worst case of 327 hours [15].

2.2.5 Physical Acquisition Techniques

The term physical acquisition, when applied to digital forensics, is an acquisition method that is able to retrieve every bit of a storage medium, without being restricted to the view presented by e.g., the file system. There are three accepted physical acquisition methods in practice today. In order of least invasive to most invasive these methods include: Flasher tools, JTAG pins, and chip desoldering. Note, these methods require special tools and are not used in this thesis. They are provided here for reference only. A pseudo-physical acquisition technique will be used in this study through the Android SDK, covered in Section 2.1.2.

Flasher Tools

Device manufacturers or service centers create flasher tools to provide debugging and diagnostics of corrupted flash devices. Flasher tools provide a hardware interface with embedded software that copies all flash memory data from the target system. In his 2007 paper [16], Al-Zarouni describes two categories of flasher boxes. The first category is "Branded boxes:" more expensive, serialized, recognized by cellular manufacturers and widely used by service technicians. The second category he terms "Unbranded boxes." He describes unbranded boxes as cheaper, can match Branded box functionality and many times ships without required software or hardware [16].

Al-Hajri claims that Flasher boxes provide more access to a mobile device's memory than command line tools, discussed in the next section. His theory is based on the fact that most command line tools utilize the device's embedded command and response protocols, like the modem-based AT protocol. He claims these commands can query the phone, but do not have direct access to the data as they are relegated to using the phone's file system. Al-Zarouni does caution against the use of Flasher boxes in an investigative process. He claims that Flasher boxes are designed to write to the phone's memory, as well as read the memory. Without fully knowing what the box is doing, the process might not provide forensically sound data. He also points out that with some boxes, vital memory areas might be skipped due to the phone's internal system and that some flasher software will skip spare areas of memory [16].

JTAG

A second method to obtain a physical acquisition is through the IEEE standard 1149.1-2001 Standard Test Access Port and Boundary-Scan Architecture, more informally known as the Joint Test Action Group (JTAG) access port. The IEEE standard describes the ports as "circuitry that may be built into an integrated circuit to assist in the test, maintenance, and support of assembled printed circuit boards" [17]. The standard calls for a standard interface to communicate with external testing tools.

The IEEE standards indicate that a general-purpose port, labeled the Test Access Port (TAP) provide test support functions to external components. The TAP has three to four input connections and one output connection. The test clock input (TCK), provides the clock for the test logic. The test mode select (TMS) receives codes and passes to the TAP controller for control test operations. The test data input (TDI) receives serial test instructions and data. The test-reset input (TRST) is optional and provides asynchronous initializations of the TAP controller.

Finally, the test data output (TDO) is the serial output for test instructions and data from the test logic [17].

In their 2007 paper, Breeuwsma Et al. experimented with flash data recovery utilizing the JTAG ports. They show that flash memory chips are not JTAG enabled, but if connected to a JTAG enabled processor, then data can be retrieved with this method. If the processor provides an external test mode, then the JTAG controller could be allowed to control all processor pins and ultimately capture a complete image of the flash chip [18]. Note that JTAG support is device dependent and not required by manufacturers.

Chip Desoldering

The most invasive acquisition technique requires physically removing the memory chip from the circuit board and recovering the memory with an external reader. There are two commonly used methods to attach a chip to the printed circuit board and achieve low profiles needed for cell phone designs. The first method, the thin small-outline array (TSOP) method employs a chip with the pin leads on the side of the chip. The second method, the Ball Grid Array method solders tiny metal balls onto the bottom of the chip that acts as the pin leads.

Breeuwsma Et al. [18] show that removing TSOP or BGA chips can be accomplished with the use of hot air. The hot air removes the soldered connections without damaging the chips and without the need for a soldering iron and a vacuum air gripper removes the chip. To correctly read from the TSOP pins, the pins require cleansing to remove old solder. The BGA will require reballing or pogo pins to correct for the different ball sizes. With the clean chip, it can now be analyzed with a reader. The study concludes that this method provides the most complete forensic image for storage. However, they also stress the risk of damage to the chip and breach of the embedded system to retrieve the chip [18].

2.2.6 Logical Acquisition Techniques

Logical acquisition is a software-based method that retrieves data based on the file system hierarchy. This method is reliable, quick, and easy to implement. A logical acquisition of a storage medium will provide data through the file system, but will disregard data the operating system deleted although the data remains on the medium.

There are multiple methods and tools to obtain a logical acquisition of a cellular device. Here we will briefly discuss proprietary solutions, an SD card solution and a solution that uses the Android SDK tool, the `adb shell`. These methods are covered to provide a thorough under-

standing of available logical solutions. This study will utilize an Android Market application to provide logical images and the Android SDK will provide images that contain allocated and non-allocated data.

Commercial Tools

The popularity of cellular devices has propelled multiple vendors into the mobile forensic market. Arguably, the three most popular vendors at the time of this writing are Paraben, Cellbrite, and Encase, all of whom provide solutions to Smartphone forensics. Commercial mobile forensic tools do provide for logical acquisition of thousands of cell phones, while some tools advertise physical acquisition for only a few devices. The data identified in Table 2.3, is a typical list of personal data commercial tools advertise their methods can extract from cellular devices.

SMS Messages
Phonebook
Call History
Datebook
Scheduler
Calendar
To-Do Lists
GPS Information
Email

Table 2.3: Typical Commercial Tool Data Extraction Capabilities

The NIST Computer Forensic Tool Testing Program tests some forensic tools. NIST performs data object extraction tests and tests for data integrity on multiple mobile devices for each tool tested. NIST publishes a final report that provides a summary of the test results and most importantly what tests the tool failed. These reports are also sent to vendors to provide software patches to their tools. The itemized NIST reports can be found at [6].

In their testing, NIST categorizes mobile phone data storage into flash read only memory (ROM) and random access memory (RAM). Flash is an EEPROM type of memory and is extensively used in Smartphones for internal storage. Flash ROM contains the pre-loaded OS and applications and stores user data. RAM is further classified as program memory and object store. The program memory region is volatile storage while the object store is nonvolatile. NIST tests a forensic tool's capability by populating memory and the SIM card with pre-defined datasets and then running the tool. Tools are tested against "required" and "optional" digital evidence.

Evidence	Location
International Mobile Equipment Identifier (IMEI)	GSM Device memory
Mobile Equipment Identifier (MEID)	CDMA Device memory
Service Provider Name (SPN)	SIM memory
Integrated Circuit Card Identifier (ICCID)	SIM memory
International Mobile Subscriber Identity (IMSI)	SIM memory
Mobile Subscriber International ISDN Number (MSISDN)	SIM memory
Personal Information Management (PIM) data	Device memory
Abbreviated Dialing Numbers (ADNs)	SIM memory
Application Data	Device memory
Internet Data	Device memory
Call logs - Incoming and outgoing calls	Device memory
Last Numbers Dialed (LND)	SIM memory
Text messages (SMS, EMS)	Device memory, SIM memory
Multi-media Messages (MMS)/email	Device memory
File storage	SIM memory
GPS related data - Longitude and latitude coordinates	Device memory

Table 2.4: NIST required digital evidence and evidence location.

Required evidence is data that are available across all handsets. The NIST's required digital evidence and where the evidence resides are found in Table 2.4 [19]. Further test cases can be found in Chapter 2 of the reports found in [6].

Mobile Internal Acquisition Tool

Currently, there exists no standard for hardware interfaces or a standard communication protocol to interact with a mobile device. Vendors issue proprietary cables, protocols, and operating systems. To offset the frustration mobile forensic examiners are facing, in 2008, Me and Rossi proposed a new mobile acquisition method. Their method involves executing a data acquisition tool from a removable SD memory card. The authors turn off the mobile device, remove the SIM card and SD card. They load their code onto a new SD card. The new SIM card and new memory card are inserted into the phone. The device is then booted in safe mode so only core system processes are spawned. Finally, the algorithm loaded on the SD card iteratively searches the file system tree, opens a file in read-only mode, computes an MD5 hash and then copies the data to the memory card. Their method shows results comparable to a well known commercial acquisition tool. The method was compiled with standard Symbian APIs to access the file system and only tested on Symbian mobile devices [20].

Later in 2008, Me and Distefano provided an assessment of the memory card acquisition scheme, now dubbed the Mobile Internal Acquisition Tool (MIAT). The paper points out that the source code must be manually compiled for each device and that the tool cannot open "locked" files residing on the SIM, which might be of concern to forensic examiners. However, their experiments showed that the MIAT performed just as well as the proprietary Paraben device seizure suite for data acquisition, but proved to be much slower [21].

Android SDK

The Google Android Operating System provides a Software Development Kit (SDK) to potential application developers. The SDK provides elective platform tools that allow developers to communicate with an Android device via USB using their proprietary Android Debug Bridge (adb) tool. The `adb` tool is a client-server program that also includes a daemon running in the background for each device instance [22].

When invoked with the "shell" option, `adb` runs the ash command shell on the attached device. The `ash shell` provides access to basic UNIX commands, including the `dd` tool. `dd` provides low-level copying of data and can convert any file and produce a new raw data file. Android mounts its partitions at `/dev/mtd` and, depending on the device, can have up to six partitions. `dd` can copy the raw data of these pseudo-devices and output the binary to the sdcard for extraction via `adb pull` or by mounting the sdcard on the examiner's machine. The benefit of using this method is that the `dd` tool will produce a bit-for-bit replica of the file system, including allocation tables, and deleted, but still resident, files in the flash partition.

An alternative method is using the `adb pull` function. This method allows a forensic examiner to directly copy the `mtd` partition to the examiner's computer. However, depending on the rooting method, the `adb pull` method requires the examiner to change the file system permissions of the `mtd` partitions under the `/dev/mtd` directory to allow the `adb shell` to pull the files.

The disadvantage to these two methods is that the phone must be "rooted" to access these files. Rooting is the process where a phone's operating system is exploited to give an ordinary user administrative privileges. The exploit varies by different OS types and versions and can include a simple application addition to the phone or flashing a new boot image to the device. Rooting may also leave the phone vulnerable to other attacks.

The Android SDK provides tools that allows examiners access to the device that doesn't require rooting the device. The previously mentioned `adb pull` command will pull certain files from

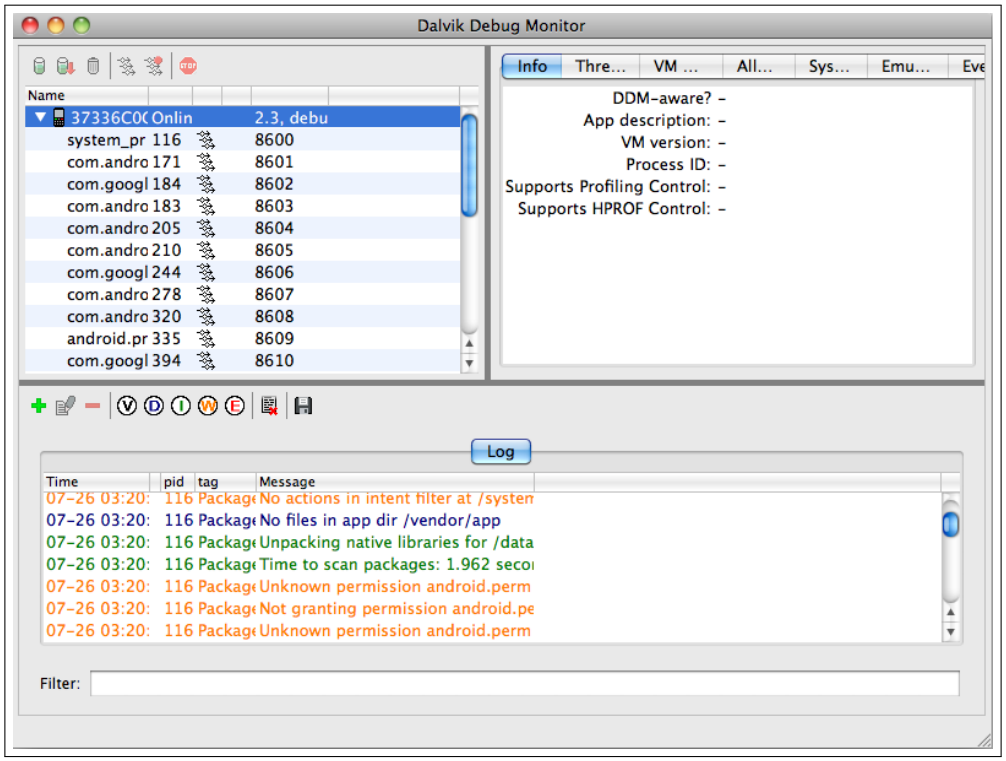


Figure 2.1: Android SDK ddms GUI.

the Android device. The `ddms` application provides a graphical user interface (GUI) to explore files on the device. Launch `ddms` from the SDK directory, attach the Android phone to the computer running `ddms`, select the device under the name panel as shown in Figure 2.1. Selecting Device, File Explorer and the application will allow the user to search files and pull files from the GUI. The `ddms` application will also allow the user to dump the device state to a text file all of the volatile data that resides on the device. The disadvantage to these techniques is that not all files are available to pull from the device.

Nandroid Backup

The Android Market contains many applications that provide backup functionality. A notable application that provides a logical acquisition technique is the Clockworkmod application available in the Android Market. The application loads a `ROM Manager` to the Android device. Through the application, a custom recovery image is downloaded and flashed to the device. Once the new recovery image is downloaded, it can be used to perform a logical backup of the boot, cache, data, recovery and system partitions on the Android device. The backups are stored on the device's SD card [23]. Obviously, one concern is the integrity of the data on the phone.

However, generally an Android device does not write data to the `recovery.img` partition, and if the examiner uses a different SD card than what was originally attached to the device the few data will be lost or overwritten. Due to the use of custom recovery images, the `ROM Manager` is not available for use on all Android devices. As the Android Market is constantly growing, newer applications may provide increased functionality and provide service to more Android devices.

2.3 Network Forensics

Network Forensics examines raw network traffic packets as they traverse the network, and analyzes network logs after an incident has occurred. The objective is to piece digital clues regarding an incident that may or may not be located on a compromised system. The objective may be to verify who was logged on during an incident, determine the origin of malicious packets or verify that a particular workstation was compromised during a breach. As with most digital data, network traffic contains metadata that identifies unique aspects of traveling data. This thesis applies Network Forensics towards the retrieval of *residual* network metadata that may be found on an Android cellular device. Packetized network traffic contains metadata in the form of IP addresses, MAC addresses and base station identifying information. Cellular devices have multiple networking interfaces to communicate with external networks. The most prevalent hardware interfaces that deliver and receive pertinent network metadata include the cellular radio and the 802.11 radios.

2.3.1 Cellular Networks

A cellular network consists of hierarchy of transceivers, controllers, and gateways to provide telephony and data services to mobile devices. The following components provide network services to a mobile device; the Mobile Switching Center (MSC), the Base Station Controller (BSC) and the Base Station (BS). The remaining elements are primarily administrative and can be reviewed at [24].

The MSC authorizes mobile devices onto the network and is responsible for call setup, teardown and call handoffs between BSCs. The MSC controls a number of BSCs and can provide Internet gateway services.

The BSC allocates radio frequencies to BSs, performs paging and performs handoffs within its BS domain. The BSC receives telephony service from the MSC and is usually connected to an Internet gateway to provide Layer 3 service to the mobile device.

The BS consists of a transceiver that delivers and receives communication from cellular devices. A BS sends beacons on periodic intervals to announce services to cellular devices. These beacons contain Layer 2 information specific to that base station, along with available Layer 3 services. The Layer 2 information of immediate interest in this thesis is the Mobile Country Code (MCC), Mobile Network Code (MCC), Local Area Code (LAC) and the base station ID. Collectively, these four items identify the physical location of the base station.

The International Telecommunication Union identifies the MCC and MNC for each country [25]. The MCC is a three digit identifier that correlates to specific countries. The MNC is a two or three digit identifier that distinguishes a cellular service provider within the country identified in the MCC. The CID relates to a unique base station and the LAC identifies a cluster of base stations.

A Smartphone's IP address remains unchanged throughout different cellular network attachments as described in Mobile IP. Mobile IP is a complex protocol. The reader is encouraged to review RFC 3344 for in-depth understanding of Mobile IP, as we will only describe the main features [26]. Mobile IP has the following main features:

- Mobile Node: Cellular device that changes it point of attachment from one network to another.
- Home Agent: A router on mobile node's home network that tunnels traffic to mobile node when not on home network. Also maintains mobile nodes current network location.
- Foreign Agent: A router on Foreign network that mobile node is attached.
- Care-of-Address: Usually the foreign agent IP address to deliver traffic to mobile node.

The Smartphone's Home Agent is usually the MSC, or equivalent, located at its home network. When on a different network, the Foreign Agent functions are performed by the MSC, or BSC. When the device attaches to a foreign network, the device sends a Care-of-Address notification to the Home Agent. When traffic arrives at the Home Agent for the mobile node, the Home Agent tunnels the traffic to the Care-of-Address for delivery to the mobile node. In the opposite direction, normal IP routing is utilized. This allows a mobile device to keep its original IP address when using the cellular radio [27].

2.3.2 IEEE 802.11

The IEEE 802.11 wireless LAN standard, commonly known as WiFi, provides technical guidance for the Media Access Control and Physical Layer protocols. The standard provides 11 channels in the 2.4 GHz frequency spectrum. 802.11 provides multiple services, including association and authentication. We provide an overview of 802.11 association here, the rest of the standard is beyond the scope of this thesis.

Association is the "service used to establish access point/station (AP/STA) mapping and enable STA invocation of the distribution system services (DSSs)" [28]. The wireless AP periodically broadcasts a beacon frame, which includes the AP's SSID and MAC address. The SSID is a set of characters that identifies the AP. The broadcast is sent on a designated channel out of the 11 channels available. Note: 11 channels are used in the U.S. and Europe; different channels are used in Japan. A mobile node scans the 11 channels looking for beacon frames. After a mobile node scans the channel spectrum, it displays the detected set of SSID's for the user to choose. When the user selects a wireless AP an associate request frame is sent from the mobile node to the AP. The AP will send an association response frame. The mobile node will then send out a DHCP discovery message to obtain a local IP address. The provided IP address will only be operational when the mobile node is within the AP's basic service set range. The mobile node's host OS will save the AP's MAC address and SSID. When the device is in reach of the AP, the mobile device will automatically associate to the AP.

2.3.3 IEEE 802.15

The IEEE Wireless Personal Access Network (WPAN) 802.15 standard is derived from the Bluetooth v1.1 specifications [29]. Therefore, Bluetooth referenced in this document will refer to the Bluetooth and WPAN standards. The Bluetooth standard transmits an omnidirectional signal in the 2.4GHz frequency to devices within 10 meters. When two or more devices are within the WPAN, they form a piconet, where one serves as a master and one or more are slaves. To reduce interference with other networks operating in this band, Bluetooth employs a frequency-hopping transceiver. The transceiver uses binary frequency shift keying and a time division duplex (TDD) scheme to enable full duplex communication. A more detailed analysis of the Bluetooth physical layer is provided in [29].

The Bluetooth TDD scheme provides slotted channels that transmit one Bluetooth packet every 625 microseconds. The packet has an access code, header and payload. The access code is 72 bits and is comprised of a preamble, sync word and a trailer. The access code allows syn-

Scanner Name	Recognizes
scan_net	Ethernet, IP and TCP packets
scan_email	RFC822 headers, hostnames, email addresses, URLs

Table 2.5: *bulk_extractor* scanners and functions.

chronization and identification between devices. The 18-bit header provides link control data in six fields. The AM_ADDR field distinguishes active members on the piconet. The TYPE field identifies the type of packet. The FLOW field indicates whether to stop or go in a asynchronous connectionless link. The ARQN field provides an ACK to the source. The SEQN bit provides sequence numbers to order a data stream. Finally, the header-error-check (HEC) provides header integrity [29].

The 802.15 protocol is similar to the 802.11 protocol. The main difference is the use of a link key to pair two Bluetooth objects. When two objects try to authenticate for the first time, they must pair. One Bluetooth object will send a random number to the second object. Once the operator of the receiving Bluetooth devices accepts, a link key is maintained between the two devices. The link is typically encrypted, using the random number as a seed for the encryption key.

2.3.4 Network Carving

In 2010, Garfinkel created the stream-based forensic tool *bulk_extractor*. Garfinkel describes stream-based forensics as the processing of an entire image as a single bytestream, starting at the beginning and reading until the end [30] . *bulk_extractor* has a modular design that incorporates a set of basic scanners, recursive scanners and optional scanners. A few network relevant scanners are shown in Table 2.5. The source code, containing more information, can be found at [31]. *bulk_extractor* modules exist to find ASCII IP address and domain names within both allocated and unallocated areas of carved images. When the net scanner is enabled, it can also carve binary Ethernet and IP address structures among multiple other data objects.

Beverly Et al. used *bulk_extractor* in their 2011 paper, [32]. The authors showed that Windows, Linux and OS X operating systems store residual binary network data structures in non-volatile storage. In their work, they force the OS into hibernation and carve for network data structures with *bulk_extractor*. Their work showed there was sufficient network forensic evidence on digital media to determine prior network connections [32].

CHAPTER 3:
Building the Smartphone Corpus

The forensic community lacks available standardized corpora to test for network data structures on Android Smartphone devices. Garfinkel Et al. stress the importance of such standardized data sets; openly available data sets enhance digital forensics through reproducibility, education, tool testing, and research [33]. This Chapter details the testing environment and hardware used for our environment (§3.1), the Smartphones under testing (§3.2), and the experimental procedure and construction of an Android Smartphone corpus (§3.3).

3.1 Test Environment

Determining the existence and extent of residual network data on Android Smartphones requires, for multiple phones, an image of non-volatile storage and an understanding of the device's communications.

As part of the larger investigation of using forensics to infer the network connectivity of Smartphones, this thesis presents a corpus of Android images in an environment with controlled Layer-2 and Layer-3 connections. For example, the corpus contains images of Android devices with data transfers over a TCP/IP and Bluetooth connection to identify interesting network metadata that persists on the device. Known ground-truth network connections provide the basis for finding discriminating markers to subsequently infer the network behavior of images of unknown provenance. The images may be downloaded from the digitalcorpora.org website at `http://digitalcorpora.org/phones/nps/`

To create known ground-truth network connections, a controlled environment was created inside a Faraday cage as shown in Figure 3.1. The environment employs a cellular GSM wireless network, two 802.11 wireless networks, and a Bluetooth storage device. More specifically, the components of the test are a Tactical base station router, Cisco access point, web server, D-Link access point, a Macbook Pro, a network packet capture tool and the Android Smartphones which are described in detail below. Because each of these network devices are under our control in our experiments, we can similarly control an Android device's association and communication with the network attachment points. Any resulting residual evidence of the network devices is then of known origin.

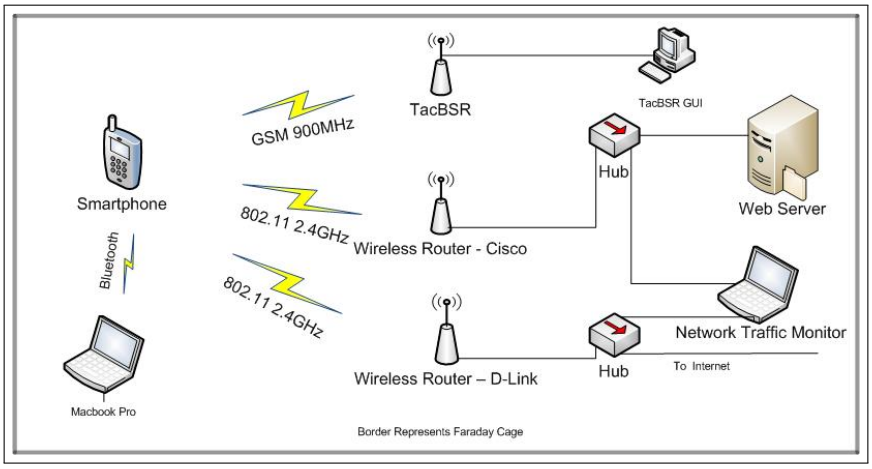


Figure 3.1: Experiment Set-up.

3.1.1 Faraday Cage

All images are created inside a Faraday cage to ensure that the phone encounters only known and controlled network radio frequencies (RF). A Faraday cage is a physical enclosure surrounded by material that blocks external electromagnetic signals from penetrating the cage and internal electromagnetic signals from leaving the cage. Our Faraday cage is surrounded by copper mesh that has been empirically tested to block terrestrial network provider's radio frequencies. The use of a Faraday cage will ensure that our cellular devices will not be corrupted with external wireless router and cellular base station metadata — thereby maintaining the integrity of the experiment. Our Faraday cage is shown in Figure 3.2.

An Anritsu MS2721A spectrum analyzer (SA) verified that the Faraday Cage was free from external GSM (850MHz, 1900MHz) or 2.4GHz signals. The SA measures the amplitude and frequency of the electromagnetic spectrum within a specified frequency range. The SA can determine if the amplitude exceeds a threshold and audibly alert the operator. The SA determined that cellular signals in the 850MHz and 1900MHz bands were not penetrating the cage. Due to the signal range and strength, during the experiments the SA was centered on 2.458 GHz with a span of 66 MHz. The 802.11 standard claims the 2.4GHz band. The SA detected 802.11 with the Faraday cage door open and provided frequent audible alarms. Once the Faraday cage door was closed, with the SA inside, the SA alarms went silent and the display showed all amplitudes receding below the threshold. The analyzer continued to run during experiments to ensure integrity.

(a) Door. (b) Workbench.

Figure 3.2: Faraday Cage.

3.1.2 TacBSR

The Tactical Base Station Router (TacBSR), manufactured by LGS Bell Labs Innovations, provides the cellular base station functionality. Manufactured to provide deployable cellular networks to military and other U.S. government entities, the TacBSR provides service in all four GSM service frequency bands and can combine multiple cellular networks. The TacBSR provides two association methods; pre-provisioned and Search and Rescue (SAR) mode. The pre-provisioned method provides network services to GSM devices that contain a SIM card that is on the TacBSR's access control list. The SAR setting allows any GSM based handset to access the cellular network through the TacBSR. This setting is intended to give accident victims cellular network access to SAR responders [34].

NPS-owned SIM cards are provisioned on the TacBSR allowing an Android phone to use the system. Each SIM card has an IMSI of 915050000000XXX, where the final three digits are the dialed number. The TacBSR is configured with the network information detailed in Chapter Two and are itemized in Table 3.1.

3.1.3 WiFi Router

We use a Cisco Linksys E2000 wireless router to provide controlled 802.11n WiFi connectivity to the Smartphone under test. Its primary function is to provide wireless services to access

RF Band:	E900Mhz
Cell ID:	2
MCC:	915
MNC:	05
LAC:	2

Table 3.1: GSM Base Station Network Settings.

the internal network as depicted in Figure 3.1. A second wireless router provides access to an external network and the larger internet. A cat-5 copper cable leaves the Faraday cage and provides connectivity. This second router is a D-Link DI-525 802.11g device. The wireless router settings are shown in Table 3.2.

Settings	Cisco	D-Link
SSID	AndroidForensics	NPS
IP Address	192.168.1.1	192.168.0.1
NAT Range	192.168.1.140–.148	192.168.0.100–.199
MAC	C0C1C040CDAB	0011953913D2

Table 3.2: Wireless Router Network Settings.

3.1.4 Bluetooth Device

The examiner's laptop, a Macbook Pro, is used to transfer files via bluetooth to the Android Device. When associating to the Android device, an associated PIN is prompted to the examiner's computer and the Android device prompting the Android user to verify the two numbers match and accept the incoming connection. Specifics of the Macbook Pro's Bluetooth hardware are provided in Table 3.3 and the files transferred over the Bluetooth interface in Table 3.4

Address	00-26-08-bc-d4-14
Manufacturer	Broadcom
Name	Neptune

Table 3.3: Bluetooth Data for Macbook Pro.

3.1.5 Web Server

The web server runs the Ubuntu 10.04 operating system with the Linux 2.6.32 kernel. The web server provides access to three files of varying sizes. The web server's Ethernet interface is assigned the primary IP address 192.168.1.117. To uniquely identify each download and

File Name	Size
to_build_be.txt	4KB
androidarch.jpg	201KB
trailheadmap.pdf	1.1MB

Table 3.4: Files transferred via Bluetooth.

later distinguish between network connections, the web server's Ethernet is also assigned three virtual IP addresses. The web server simulates Internet connections and Internet "activity," but in a controlled environment. The file names, sizes, and their virtual IP address are delineated in Table 3.5

File Name	Size	Virtual IP Address
huge.img	10MB	192.168.77.1
large.img	1MB	192.168.77.2
small.img	10KB	192.168.77.4

Table 3.5: Files provided by file server.

3.1.6 Examiner's Computer

The external computer used to acquire data from the phone's partitions requires the Android SDK in our image techniques [22]. Methods described in Chapter Two are available, but the SDK acquisition technique is relatively well accepted and reliable. The Android SDK provides tools to communicate via USB with the Android device from the examiner's computer.

3.2 Smartphones under Test

Two Android-based Smartphones are used in the experiments, an HTC Nexus One and the Samsung Nexus S. Two phones are used to show the differences, if any, between YAFFS2 and EXT4 files systems when storing network metadata. The HTC-manufactured Nexus One was the first Android device co-developed by Google. The Nexus One uses the YAFFS2 file system and is sold unlocked, meaning it will work with any cellular provider, and permits the installation of arbitrary and unsigned applications.

The Samsung Nexus S was the second phone co-developed with Google. The Nexus S uses the EXT4 file system for non-volatile storage of the system, `userdata` and `media` partitions. However, it also uses the YAFFS2 file system in `mtd` partitions similar to the Nexus One shown in Table 3.6.

The experiments here were limited to two devices due to time considerations. Each Android OS Smartphone manufacturer has the ability to change performance factors on their device. This leads to the possibility that different manufactured devices will behave differently, which is not considered in this thesis.

dev/	size	erasesize	name
mtd0	00200000	00040000	“bootloader”
mtd1	00140000	00040000	“misc”
mtd2	00800000	00040000	“boot”
mtd3	00800000	00040000	“recovery”
mtd4	1d580000	00040000	“cache”
mtd5	00d80000	00040000	“radio”
mtd6	006c0000	00040000	“efs”

Table 3.6: Nexus S YAFFS2 mtd blocks.

3.2.1 Smartphone Configuration

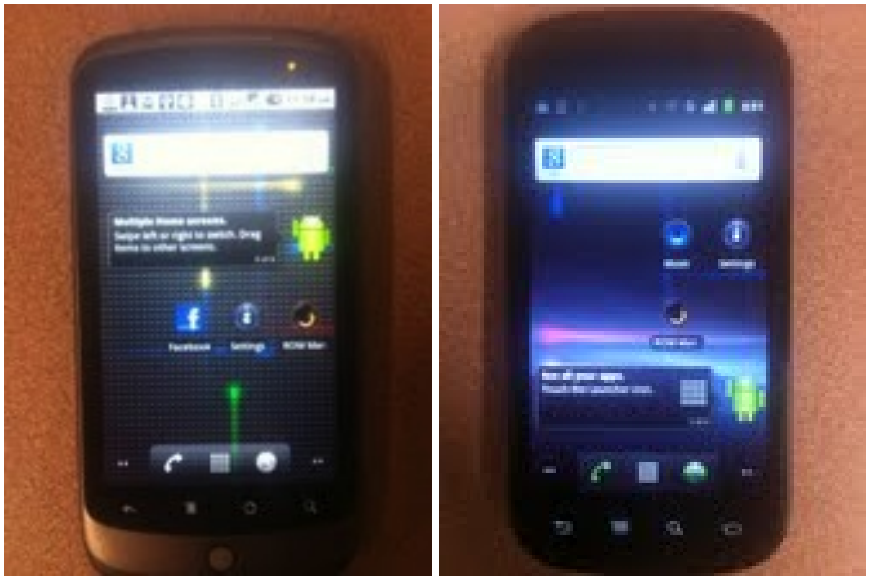
The software specifications of each device is given in Table 3.7. **VERIFY NS MAC ADDY**

Settings	Nexus S	Nexus One
Linux kernel	2.6.35.7	2.6.32.9
Android OS	2.3.1	2.3.3
Android Build	GRH78	FRG83
Processor	1GHz Cortex	1GHz Tegra 2
File System	EXT4	YAFFS2
Flash	16GB iNAND	16GB NAND
IMSI	915050000000120	915050000000122
MAC	B4:07:F9:F0:AD:77	90:21:55:2C:4D:67
Bluetooth MAC	3C:5A:37:89:40:90	90:21:55:2C:4D:67
RAM	512 MB	512 MB

Table 3.7: Smartphone configurations.

The Android OS provides a debugging protocol `adb` across a USB interface to external platforms that have the Android SDK installed. To use the `adb` protocol, the `USB debugging` mode must be enabled on the device. USB debugging is enabled under `Settings/ Applications/ Development` as shown in Figure 3.4.

To image devices at the block level, the Smartphones must be “rooted” as discussed in Chapter 2. The Nexus S is rooted using the `superboot` tool. `Superboot` flashes a customized `boot.img`



(a) Nexus One (b) Nexus S

Figure 3.3: Android Smartphones used in experiments.

file to the Android device. From the `superboot` directory in the examiner's computer, the `superboot` tool was used to unlock the bootloader of both devices with the `./fastboot-mac oem unlock` command. The Nexus S is then rooted using superboot's provided install tool, `install-superboot-mac.sh` command. `Superboot` is available for download with rooting instructions from [35].

The two devices use different Android OS versions. Rooting techniques depend on exploitations found in each version, so the two devices require different rooting applications. The Nexus One is rooted with the `Gingerbreak.exe` application. From the examiner's computer, `Gingerbreak` installs itself on the Android Device which is tethered via USB to the examiners computer. The `Gingerbreak` application on the device is selected, prompting the user to confirm the rooting action. The `Gingerbreak` application is available from [36].

The logical acquisition technique we employ utilizes the `ROM Manager` application, available from the Android market. The application will provide logical dumps of the file systems to analyze and compare to the physical dumps the Android SDK provides. The download installs the `ROM Manager` application onto the device. To perform logical backups, the application needs to flash a custom recovery image to the Android device. Under the application's main screen, select `Flash ClockworkMod Recovery`, and the application prompts the user to select the correct device. The ROM Manager downloads and flashes a recovery image for the specific

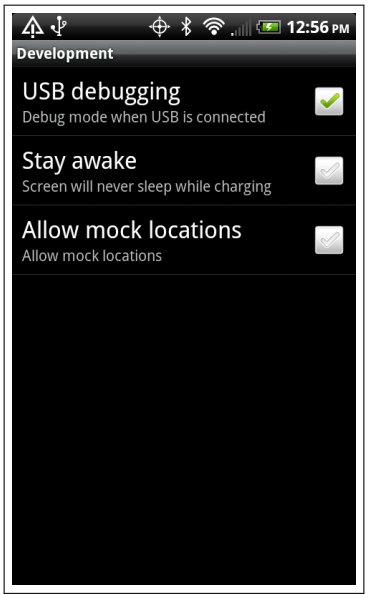
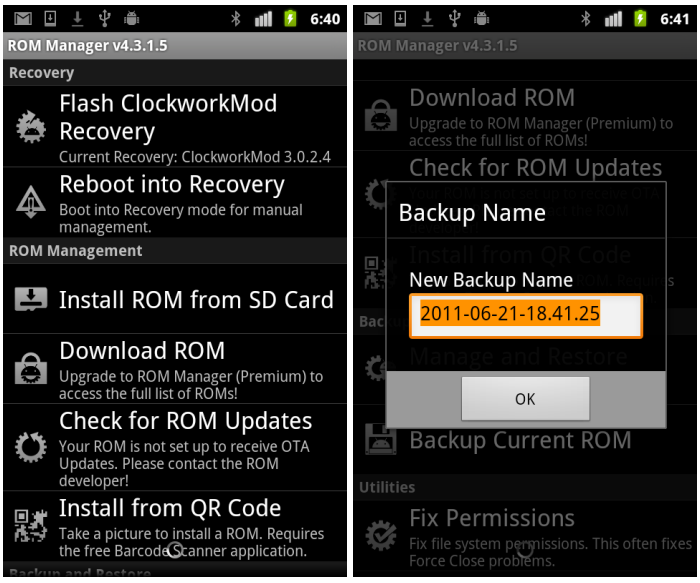


Figure 3.4: The USB Debugging Option.

Android device. The ROM Manager main screen is show in 3.5. To perform the backup, the `Backup Current ROM` is selected. The prompt to conduct a backup is show in 3.5.



(a) Main screen. (b) Backup prompt.

Figure 3.5: ROM Manager screen shots.

3.2.2 Associating a Google Account

One of the benefits of using an Android Smartphone is associating a Google Gmail account, which provides seamless email pushing to the mobile device and availability to use the Gmail account to interact with other Google applications. Google has a rich set of applications familiar with most users which can enhance everyday activities like using its calendar, Google Earth or even Google Latitude to search for friends. As the applications run, they are constantly streaming data to and from Google servers in the background. When the Google account is initially associated to the Smartphone, user data is synced to the phone and other packages are pushed to the device. One of particular interest is the Google Location package, which will instantiate the `cache.cell` and `cache.wifi` files after the association is complete. For our experiments, we associated the previously created `nps.forensics@gmail.com` account.

3.2.3 Imaging Techniques

Each mobile device was imaged inside the Faraday cage prior to conducting the experiments. The Nexus S was a newly acquired phone which was briefly used on the campus network. Its use was limited to downloading and testing the ROM Manager on the device. The NPS owned Nexus One has prior limited use with student experiments. We therefore provide a close as possible virgin, baseline copy of data on the phone's flash memory by flashing an unused stock `userdata.img` file to the device prior to each experiment, outlined in the data wiping section below.

A `Clockwordmod ROM Manager` backup is performed first, which provides a logical acquisition of the device. The following steps perform the backup:

1. Launch `ROM Manager`.
2. Select backup ROM.
3. The ROM Manager will prompt a data and timestamp of the backup as shown in 3.5. Select OK.

The Nexus S device is then imaged using the adb pull command:

1. `adb pull /dev/mtd/mtd0ro`
2. `adb pull /dev/mtd/mtd1ro`

3. `adb pull /dev/mtd/mtd2ro`
4. `adb pull /dev/mtd/mtd3ro`
5. `adb pull /dev/mtd/mtd4ro`
6. `adb pull /dev/mtd/mtd5ro`
7. `adb pull /dev/mtd/mtd6ro`
8. `adb pull /dev/block/mmcblk0`

The Nexus One rooting technique does not allow direct use of the `adb pull` command. To gain superuser access, the `adb shell` must be initiated, and root access gained (*su*). In contrast, the rooting technique for the Nexus S gave the adb shell superuser status without user input. Therefore, the access permissions of each `mtd` block must be changed, using `chmod`, prior to performming an `adb pull`. Specifically, the device is attached to the examiner's computer via USB, then:

1. `adb shell`. *Initiates ash shell on Android device.*
2. `su`. *Provides superuser status to shell.*
3. `chmod 777 /dev/mtd/mtd0ro`.
4. `chmod 777 /dev/mtd/mtd1ro`
5. `chmod 777 /dev/mtd/mtd2ro`
6. `chmod 777 /dev/mtd/mtd3ro`
7. `chmod 777 /dev/mtd/mtd4ro`
8. `chmod 777 /dev/mtd/mtd5ro`
9. `exit`. *Leaves the superuser status.*
10. `exit`. *Quits the shell session.*
11. `adb pull /dev/mtd/mtd0ro`

12. `adb pull /dev/mtd/mtd1ro`
13. `adb pull /dev/mtd/mtd2ro`
14. `adb pull /dev/mtd/mtd3ro`
15. `adb pull /dev/mtd/mtd4ro`
16. `adb pull /dev/mtd/mtd5ro`

3.2.4 Data wiping method

After completion of each experiment, the logical and physical images are acquired by the examiner's computer. Next, the mobile device's `userdata` partition is erased by attaching to the examiner's computer via USB and performing the commands below. The `ROM Manager` applications must be pushed back to the device from the SD card or the examiner's computer.

Empirical tests have shown that the Factory Data Reset will remove data from the userdata partition. To ensure the data doesn't persist between experiments, we will flash an unused `userdata.img` file thereby overwriting the current `userdata` partition. The `userdata.img` file is a stock image file that is used to overwrite the existing userdata partition. The file contains initial data to inform the file system that it's the start of the userdata partition, and is then filled with 0xFF values. The file is available from [37]. Zeroes are added to the `userdata.img` file to create a file large enough to fill the partition to overwrite any lingering data from previous experiments.

The following procedure is used on the Nexus S Android device:

1. Select settings.
2. Select Privacy.
3. Select Factory Data Reset, Figure 3.6
4. Select Reset Phone.
5. Select Erase everything.

The following actions are taken on the examiner's computer:

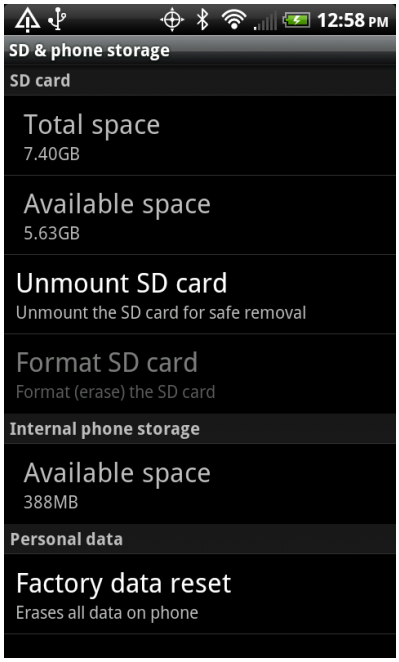


Figure 3.6: Android Factory Data Reset Option..

1. `fastboot erase userdata`.
2. *Nexus One Only*. `fastboot flash userdata userdata2.img`.
3. *Nexus One Only*. `fastboot erase userdata`.
4. `fastboot flash userdata userdata.img`.
5. `fastboot reboot`.

The ROM Manager application was stored on the Nexus One's SD card. Since the Factory Data Reset removes all user installed applications, it must be reinstated after every reset. The following commands are used for readying the Nexus One:

1. `adb shell`
2. `su`
3. `cd /sdcard`
4. `dd if=com.koushikdutta.rommanager-1.apk`
   `of=/data/app/com.koushikdutta.rommanager-1.apk`

5. `chmod 644 /data/app/com.koushikdutta.rommanager-1.apk`
6. `reboot`

The Nexus S `ROM Manager` application has to be pushed to the device after each Factory Data Reset. The following commands are used for the readying the Nexus S:

1. `adb push com.koushikdutta.rommanager-1.apk /data/app/com.koushikdutta.rommanager-1.apk`
2. `adb shell reboot`

3.2.5 Commercial Application Data Storage

We include an experiment with the popular Facebook Android application to explore how such applications store network data, if any. A Facebook account with the email address `nps.forensics@gmail.com` and user name `NPS Forensics` is used for Facebook credentials. The account does not have any friends and is used to post status updates and pictures. While the Nexus One has the Facebook application pre-installed, the Nexus S requires the application to be pushed from the command prompt. For the Nexus S experiments containing Facebook access, the following action is taken at the examiner's computer after the `ROM Manager` application is pushed to install the Facebook for Android 1.2 application to the Nexus S:

1. `adb push facebook_for_android_1.2.apk /data/app/facebook_for_android_1.2.apk`

3.3 Images

Table 3.8 is an overview of the images provided by this thesis. The experiments are ordered by smallest to largest data footprint each experiment will have on flash memory. Although a Factory Data Reset was performed along with overwriting the user data partition with zeros, this order helps ensure that older files are overwritten by new data.

The following experiments provide an in-depth examination of flash memory after the Android device transfers data through different network interfaces. Two network interfaces will conduct data transfers; the wireless 802.11 interface and the Bluetooth 802.15 interface. The stock web

Name	Summary
2011-NexusX-1	Pseudo-factory reset
2011-NexusX-1.5	Google account association
2011-NexusX-2	Network beacon metadata
2011-NexusX-3	Bluetooth metadata
2011-NexusX-4	Small file transfer over IP
2011-NexusX-5	Large file transfer over IP
2011-NexusX-6	Facebook application use
2011-NexusX-7	Combined with telephony

Table 3.8: Images created for study.

browser and a downloaded application will transfer data over 802.11. The popular Facebook application provides an instance to study how applications store network data within the flash data partition.

Note, all experiments are preceded by accessing an external network with Internet access. The ROM Manager requires Internet access to download the recovery image that provides the logical image functionality. To mitigate contaminating the device with network data, the network traffic to and from the device was captured with the Wireshark traffic monitoring tool. The pcap output file provides data that is verified against the resulting images, and accounts for all traffic to and from the device.

The following list of actions are performed prior to each experiment with the exception of 2011-NexusX-1.

1. Power on D-Link router.
2. *On Android device:* `select Settings.`
3. `Select Wireless & Networks.`
4. `Select Wi-Fi Settings.`
5. `Select Wi-Fi Turn On.`
6. `Select NPS, then select connect.`
7. `Select Settings.`

8. Select Accounts & sync.
9. Enter Google Account. Our account is nps.forensics@gmail.com.
10. Initiate ROM Manager application.
11. Select Flash ClockWorkMod Recovery.

3.3.1 2011-NexusX-1

This experiment provides the baseline Factory Data Reset and flashing operation baseline to wipe user data. The baseline includes the device associating to the D-Link router to get the recovery image, but does not associate a google account to the device.

1. Wipe Android device in accordance with erasing methods delineated above.
2. Image Android device.

3.3.2 2011-NexusX-1.5

This will determine how associating a google account to an Android device will effect the device. This image is processed based on the list under the Experiment section, which associates a google account and downloads the ROM Manager recovery image.

3.3.3 2011-NexusX-2

This experiment will determine how the cellular device retains network metadata based off associating to network nodes and processing wireless beacons.

1. Power on Cisco Router.
2. On Android device: select settings.
3. Select Wireless Networks.
4. Select Wi-Fi Settings.
5. Turn on Wi-Fi radio.
6. Select AndroidForensics.

7. `Select connect.`
8. Power on TacBSR.
9. To ensure device scans TacBSR.
10. `Select Settings.`
11. `Select Wireless & Networks.`
12. `Select Mobile Networks.`
13. `Select Network Operators.`
14. `Select Search Networks.`
15. Power on D-Link Router.
16. Let run for 1 minute.
17. Power off D-Link.
18. Power off TacBSR.
19. Power off cisco router.
20. Image Android Device.

3.3.4 2011-NexusX-3

Associating with Bluetooth Devices This experiment show how Bluetooth network metadata persists on the Android device. Note that the Nexus One saves Bluetooth transferred files to the SD card, which is not imaged as part of these experiments.

1. On Android device: `select Settings.`
2. `Select Wireless & networks.`
3. `Select Bluetooth settings.`
4. `Enable Bluetooth.`

5. On the Macbook: `Power on Bluetooth radio.`
6. On the Macbook: `Enable Bluetooth sharing.`
7. Pair Bluetooth device to Android device.
8. On the Macbook: `Send to_build_be file.`
9. On Android device: `Pull down notifications list and accept incoming file.`
10. On the Macbook: `Send androidArch.jpg file.`
11. On Android device: `Pull down notifications list and accept incoming file.`
12. On the Macbook: `Send trailheadmap.pdf file.`
13. On Android device: `Pull down notifications list and accept incoming file.`
14. Power off Bluetooth, Macbook.
15. Image Android Device.

3.3.5 2011-NexusX-4, 2011-NexusX- 5

This experiment will check the Android device for network data structures after downloading different files sizes from the local file server. This test is conducted for the small and large files as indicated in Table 3.5. The 2011-NexusX-4 image will download the small.img file. The 2011-NexusX-5 image will download the large.img file.

1. Power on TacBSR.
2. Power on cisco router.
3. Associate to TacBSR.
4. `Select Settings.`
5. `Select Wireless & Networks.`
6. `Select Mobile Networks.`
7. `Select Network Operators.`

8. Select Search Networks.
9. Select 91505.
10. Associate Android device to cisco router.
11. Select Wireless Networks.
12. Select Wi-Fi Settings.
13. Turn on Wi-Fi radio.
14. Select AndroidForensics.
15. Select connect.
16. On Android device open web browser.
17. Point browser to specified IP address.
18. Download file.
19. Power off TacBSR.
20. Power off cisco router.
21. Image device.

3.3.6 2011-NexusX-6

This experiment intends to show how applications store network data structures and determine how they persist on the Android device. This experiment will use the Facebook application, as it is very popular and comes installed on some Android devices.

The Nexus S did not come with Facebook pre-installed, so it was pushed from the examiners computer to the mobile device.

1. Power on D-Link router.
2. Associate device to router.
3. Select Wireless Networks.

4. `Select Wi-Fi Settings.`
5. `Turn on Wi-Fi radio.`
6. `Select NPS.`
7. `Select connect.`
8. `Open Facebook application.`
9. `Authenticate credentials to Facebook application.`
10. *Post a status update on Facebook application.*
11. *Check profile wall.*
12. *Upload first image.*
13. *Upload second photo.*
14. Power off wireless router.
15. Image Android device.

3.3.7 2011-NexusX-7

This experiment encompasses all network data structures captured in previous experiments. We will also place phone calls between two devices on the same TacBSR. The telephony data is not relevant to this thesis, but will be useful data for the public corpus.

1. Power on D-Link router.
2. Associate to D-Link router.
3. `Select Wireless Networks.`
4. `Select Wi-Fi Settings.`
5. `Select NPS.`
6. `Select connect.`

7. On Android device: `select Settings.`
8. `Select Wireless & networks.`
9. `Select Bluetooth settings.`
10. `Enable Bluetooth.`
11. On the Macbook: `Power on Bluetooth radio.`
12. On the Macbook: `Enable Bluetooth sharing.`
13. Pair Bluetooth to Macbook.
14. On the Macbook: `Send to_build_be.txt file.`
15. `Open Facebook application.` *Update profile status.*
16. Power on TacBSR.
17. On Android device: `Select Settings.`
18. `Select Wireless & Networks.`
19. `Select Mobile Networks.`
20. `Select Network Operators.`
21. `Select Search Networks.`
22. `Select 91505.`
23. Place phone call between devices. 20s.
24. `Open Facebook application.` *Upload first image.*
25. On the Macbook: `Send AndroidArch.jpg file.`
26. Power on cisco router.
27. Associate device to cisco router.
28. On Android device: `Select Wireless Networks.`

29. `Select Wi-Fi Settings.`
30. `Select AndroidForensics.`
31. `Select connect.`
32. Open web browser.
33. Point browser to specified IP address.
34. Download the huge.img file.
35. On the Macbook: `Send trailheadmap.pdf file.`
36. On Android device: `Select Wireless Networks.`
37. `Select Wi-Fi Settings.`
38. `Select NPS.`
39. `Select connect.`
40. `Open Facebook application.` *Upload second image.*
41. Power off TacBSR.
42. Power off D-Link router.
43. Power off Cisco router.
44. Image Android device.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Data Recovery

The following image analysis will test the hypothesis that the Android OS persists network metadata to flash non-volatile memory. To test this hypothesis, we will determine whether the network data structures created in Chapter 3 persist, either on currently allocated or unallocated pages of the flash file system.

4.1 Analysis Preparation

This chapter analyzes both the physical and logical images recovered from flash memory. Analyzing both is important for e.g., mapping discovered network metadata to the corresponding allocated pages within the file system, or to determine that the data is not longer current, i.e., unallocated, in the file system. Recovered images contain contain different file system partitions: the YAFFS2 flash file system and the Linux-based EXT4 file system. Therefore, for the image analysis, we use two file analysis methods. For YAFFS2, we will perform a physical examination of the YAFFS2 OOB data. For EXT4, we will mount the EXT4 file system onto a Linux machine. The two methods will provide network metadata association to files in accordance with each file system. To determine what network data structures reside on non-volatile memory, network metadata signatures are developed to search for network data structures of interest (§4.1.1).

YAFFS2 File Analysis

Regan showed that YAFFS2 stores the file ID in ASCII within the flash out-of-band (OOB) area that directly follows the data within a flash page [10]. After one of the carving tools identifies relevant network metadata (§4.1.2), that data can easily be accessed through a hex editor search function. The hex editor Synalyze It is used to conduct this analysis. Synalyze It search function provides the user with a separate window pane to review the search results. The results show the data's offset within the file and shows the previous 5 bytes leading up to the data. This provides the examiner an indication if the data is from the same file, which is manually inspected against the OOB contents. Once the data structure is found, the OOB area will identify the original file name given containing the network metadata. This provides a simple method to identify the file that created the residual network metadata.

EXT4 File Analysis

File analysis with the EXT4 file system is performed by mounting the Nexus S EXT4 partitions (system, userdata, internal SD card) with the `mount -t ext4 /filedir /mnt -o loop -r` command. Once mounted, a file can be found with the `find /mnt filename -depth` command for further analysis. The computer performing this service is a Ubuntu 11.04 with a Linux 2.6.38-10 kernel.

The EXT4 flash partitions do not contain the file metadata in OOB, therefore a different method must be used. Examining the file contents found in YAFFS2 tests showed the same files stored the network metadata in the EXT4 file system. The YAFFS2 analysis could be conducted with a Linux kernel compiled to allow a YAFFS2 file system support. The YAFFS2 analysis method employed for this thesis was provided to give a second analysis method that is quick and easy to implement.

4.1.1 Carving Signatures

IMSI

The International Mobile Subscriber Identity (IMSI) value is a 15 digit value where the first five digits are the MCC followed by the MNC. The remaining 10 digits are the device's Mobile Subscription Identification Number (MSIN), which is commonly referred to as the device's telephone number. The IMSI is stored on a Subscriber Identity Module (SIM) in GSM devices. The Android device's IMSI values can be found in §3.2.

Empirical tests have shown that the IMSI is commonly found in `userdata` partitions on Android devices coded in ASCII. Through exhaustive YAFFS2 and EXT4 file analysis techniques, we can attribute all discovered 15 digit IMSI strings on the physical image to two separate files; the `CheckinService.xml` file located under the `/data/ data/ com.google.android.gsf/ shared_prefs/` directory and from the IMSI column in the `accounts` database file located in the `/data/ system/ registered_services/` directory.

The Checkin Service is an Android service that looks for updates to the device's firmware and updates to applications that have been downloaded from the Android market. When the service is invoked, it saves multiple data including the firmware and the IMSI value on the device.

The signature developed for the *netcarver* tool identifies the string `CheckinService_lastSim` and stores the value within the following XML struct > and < brackets. The signature also searches for the string `imsi` and copies the following 15 digits into memory. When a `userdata`

partition was loaded onto *netcarver* , the tool found all instances of the IMSI that was identified through the hex editor. *netcarver* itemizes the captures by the previous 2 hexadecimal values. In each instance, there were two captures, multiple IMSI values attributed to the `CheckinService.xml` file and one instance attributed to the textttaccounts database. This only scans for IMSI numbers stored in ASCII/UTF-8.

SSID

The Service Set Identifier (SSID), initially described in §2.3.2, is the set of ASCII characters that identifies a wireless access point. Thorough YAFFS2 and EXT4 file analysis techniques prosecuted against SSID values of known provenance allows us to attribute all discovered SSID strings to the `wpa_supplicant.conf` file located under the `/data/ misc/ wifi/` directory. When a device associates to a wireless router, the `wpa_supplicant.conf` file will store the SSID and the associated pre-shared key password, if one is required.

The signature developed to identify SSIDs searches for the string "`ssid=`" and then saves the SSID that is bookended by quotation marks. This method was implemented into *netcarver* which proved to retrieve all SSID instances in the `userdata` partition. Allocated and when applicable, unallocated `wpa_supplicant.conf` instances were recovered. The *netcarver* results were verified against the instances found with the hex editor.

Cellular Base Stations

Cellular base stations transmit four locational metadata items of interest originally covered in §2.3.1. The Mobile Country Code (MCC), Mobile Network Code (MNC), Base Station Identifier (CID) and Local Area Code (LAC) identify a cellular base station to a physical location. ITU standards put MCC and MNC values as 3 and 2 digits respectively [25]. Currently, there is not an openly available source that standardizes CID and LAC values. Empirical tests have shown CIDs with up to 2 ASCII digits and LACs up to 4 ASCII digits present on Android devices.

Empirical tests have shown the google location package `cache.cell` file, located under `/data/ data/ com.google.android.location/ files` directory saves base station metadata in the MCC:MNC:CID:LAC format coded in ASCII. The cellular network metadata format was found by examining the textttadb get-state report. The `get-state` command performs a RAM dump of the connected device which includes multiple data, including CID, LAC, MNC and MCC values. Using the values found in the `get-state` report, an exhaustive YAFFS2 and EXT4

file analysis for the binary and ASCII metadata values revealed the metadata was stored by the `cache.cell` file.

The signature used to identify cellular base station metadata stored by the `cache.cell` file searches for a data structure in colon-digit notation, X:X:X:X, where each X represents up to four octets. This method produces false positives, but upon manual inspection, most false positives are easy to filter. As before, we only scan for metadata in ASCII.

MAC Addresses

Wireless access points contain a Layer 2 Media Access Control (MAC) address as articulated in §2.3.2. The canonical MAC notation is six hexadecimal bytes separated by colons, shown as 11:22:33:AA:BB:CC. MAC addresses are not duplicated so they are unique to each device. Searches are conducted for MAC addresses coded in ASCII and binary.

Our file analysis methods show that the google location package `cache.wifi` file located under the `/data/ data/ com.google.android.location/ files` directory accounts for all ASCII coded MAC address data from wireless access points. The file saves the MAC address of associated wireless access points. The file only saves the MAC address.

Bluetooth radios also have unique MAC addresses (BTADDR). Observed BTADDRs from associated devices persist on an Android device coded in ASCII and are found in multiple files. The files attributed to associated BTADDRs and the frequency of their activity is found in §4.2.3.

DHCP Acknowledgments

The Dynamic Host Configuration Protocol (DHCP) dynamically assigns IP addresses to nodes on a network [38]. When a node associates to a network, it sends a DHCP Request to the DHCP server. If all conditions are met, the DHCP server ultimately responds to the requesting node with a DHCP ACK that contains the requesting node's new IP address, the DHCP server's IP address and the requesting node's gateway IP address. The DHCP ACK payload contains the Bootstrap Protocol (BOOTP), which was the predecessor to DHCP. The BOOTP has multiple fields, which includes the IP address of the client and server. The first three BOOTP bytes are found in Table 4.1.

Multiple observations of a router's big endian binary RFC1918 IP address persisting on the Android device's `userdata` partition reveals the file responsible the IP address data begins with the hexadecimal values 0x020106. Manually inspecting the file on the Android device against the BOOTP shows that the BOOTP section of a DHCP ACK persists on the Android `userdata`

partition. File analysis shows the file preserving the data is the `dhcpcd-eth0.lease` file, located under the `/data / misc/ dhcp/` directory. When two routers are used, or the single routers lease expires we find multiple DHCP ACKs all attributed to the `dhcpcd-eth0.lease` file.

The signature method developed for DHCP ACKs identifies the first three BOOTP bytes when they follow a 0xFF byte. The 0xFF byte is the end of empty space before the new page starts at 0x020106. In first-hand experiments, when searching for 0x020106 presents hundreds of false positives. Using the 0xFF byte as a filter presented at most two false positives. Once the DHCP ACK is identified, the signature copies the nodes IP address which begins 17 bytes offset from the start of the page. The search algorithm can easily be manipulated to return the DHCP server's IP address as well which is present at the end of the DHCP ACK.

Byte	Hex value	Bootstrap Protocol
1	0x02	DHCP Boot reply
2	0x01	Hardware type: Ethernet
3	0x06	Hardware address length

Table 4.1: First three bytes of the bootstrap protocol.

Binary IP Addresses

While cellular and wireless base station identifiers provide a potential record of the physical locations of a mobile Android device, they give no indication of with whom the device communicated. One potential source of communication data is residual binary IP addresses.

To determine if binary IP addresses from the connection to the external internet and the internal web server persist on the Android device, we use the captured corpus pcap files to determine ground-truth IP addresses. To obtain a list of IP addresses to search for, we use Wireshark to filter the network capture to include only the IP address given to the D-Link router. IP packets addresses to and from the D-Link router are filtered by conversation to show the second node's IP address. The Cisco router's RFC 1918 address, the web servers IP address, the D-Link router's and Android device's RFC 1918 addresses are added to the Wireshark capture list. The *ipcarver* network carving tool, described below, examines the flash media in search of the binary IP addresses. If *ipcarver* identifies a potential IP address data structure, the data structure is analyzed with the *netcarver* carving tool described below. The *netcarver* data is presented as a table within each analysis section.

Note, the `system` partition is not overwritten or otherwise forensically cleared (wiped) after each experiment, therefore once binary IP addresses are written to that partition, they are presented in the appropriate section, then not referenced again, even if found in a different experiment. Also, binary IP addresses correlating to DHCP ACKs are not included in the binary IP tables.

Network Metadata Signature Summary

Table 4.2 summarizes the carving signatures based on the preceding analysis using *ipcarver* and *netcarver* against the corpus images for each type of network metadata considered.

Metadata	Signature
IMSI	"CheckInService_lastSim>"
SSID	"ssid="
Cellular Base Stations	MCC:MNC:CID:LAC
MAC Address	11:22:33:AA:BB:CC
DHCP ACK	0xFF020106

Table 4.2: Network metadata signatures.

4.1.2 Carving Tools

To the best of our knowledge, our work is the first to recover network metadata from Android flash memory. The *bulk_extractor* tool carves low level binary network data structures, but the tool disregards binary data structures that are not in an IP packet or network socket format [32]. The analysis in this chapter is therefore the result of using two tools: *ipcarver* and *netcarver* , a tool created specifically to extract network metadata pertinent to this thesis.

The first tool, *ipcarver* is an NPS-internal program used to perform the N-gram frequency analysis in [32] and eventually modeled the network carving modules for *bulk_extractor*. *ipcarver* performs a `grep`-like search for binary IP and MAC data structures. When a data structure is found, *ipcarver* will provide the operator feedback on whether the data structure is a valid binary IP address (based on checksum) and provide an N-gram frequency output to discriminate surrounding bytes of potential IP packets. Each experiment's network traffic capture `pcap` file provides the IP and MAC address granularity to search for specific data structures that might persist on each image.

netcarver is another network carving tool based on the *ipcarver* framework, but modularized to search for data signatures identified in §4.1.1.

4.2 Image Analysis

The images created in Chapter §3 were analyzed for network data structures using the aforementioned *ipcarver* and *netcarver* carving tools and the signatures in Table 4.2. The 2011-NexusX-1 baseline image was analyzed for data structures resulting from the required internet access to download the `ROM Manager` recovery image. The baseline images are then compared to the 2011-NexusX-1.5 images to delineate any changes. Thereafter, the 2011-NexusX-1.5 images are considered the baseline image due to each remaining experiment's requirement to associate the google account, as discussed in §3.2.2, with the Android device.

The physical image partitions remain the same size through each experiment, while the `cache` and `data` partitions fluctuate in the logical images as shown in Table 4.3.

Partition	Nexus One	Nexus S
mtd0ro	918 KB	2.1 MB
mtd1ro	4.2 MB	1.3 MB
mtd2ro	3.7 MB	8.4 MB
mtd3ro	152 MB	8.4 MB
mtd4ro	99.6 MB	492.3 MB
mtd5ro	205.8 MB	14.2 MB
mtd6ro	N/A	7.1 MB
mmcblk0p1	N/A	536.9 MB
mmcblk0p2	N/A	1.07 GB
mmcblk0p3	N/A	14.31 GB

(a) Physical images

Partition	Nexus One	Nexus S
boot	3.7 MB	8.4 MB
cache	5 — 70 MB	16 KB
data	13 — 14.8 MB	24.3 — 28.2 MB
recovery	4.2 MB	8.4 MB
system	129.2 MB	155.3 MB

(b) Logical images

Table 4.3: Experiment partition sizes.

Table 4.4 reiterates the images produced in Chapter §3. These images will be scrutinized for network metadata identified in §4.1.1. Each image contains two versions; the 2011-NexusX-Y version represents the physical acquisition method and the 2011-NexusX-Y-1 version represents the logical acquisition method. Both are described in §3.2.3.

The 2011-NexusX-7 images are presented in the Associating a Google Account analysis section because the Nexus One did not instantiate the `cache.wifi` and `cache.cell` files until the final experiment. Empirical testing shows there was no deterministic point at which these files were written to non-volatile memory. We leave a more detailed investigation to future work.

Each image will then be analyzed and presented by relevant sections:

- Baseline Image: 2011-NexusX-1
- Associating a Google Account: 2011-NexusX-1.5 and 2011-NexusX-7
- Effects of File Transfer via Bluetooth: 2011-NexusX-3
- Effects of File Transfers: 2011-NexusX-4 and 2011-NexusX-5
- Effects of Application Use: 2011-NexusX-6

Name	Summary
2011-NexusX-1	Baseline Image
2011-NexusX-1.5	Google account association
2011-NexusX-2	Network beacon metadata
2011-NexusX-3	Bluetooth metadata
2011-NexusX-4	Small file transfer over IP
2011-NexusX-5	Large file transfer over IP
2011-NexusX-6	Facebook application use
2011-NexusX-7	Combined with telephony

Table 4.4: Images created for study.

4.2.1 Baseline Image Analysis

The 2011-NexusX-1 image provides a baseline from which to examine further network data structures.

Nexus One Results:

Item	Count
ASCII IMSI (915050000000122)	15
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1
Binary IP address `192.168.0.1` in the `system` partition	51

Table 4.5: 2011-NexusOne-1 Quick Summary.

As indicated in Chapter 3, the baseline image required internet access to download the `ROM Manager` custom recovery image. The corresponding wireless router association initiated a

Item	Count
ASCII IMSI (915050000000122)	2
NPS SSID instance	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
Binary IP address 192.168.0.1 in the system partition	51

Table 4.6: 2011-NexusOne-1-1 Quick Summary.

DHCP request that prompted the D-Link wireless router to respond with a DHCP ACK. The DHCP ACK persists in the userdata partition as shown in Figure 4.1. Both images contained one instance of the binary IP address 192.168.0.100 from the DHCP lease file. Both instances contain the same four-byte Bootstrap Protocol Transaction ID number 0x25C7B67F. With a fair degree of certainty, we can conclude that these two files are the same.

The D-Link router's SSID "NPS" persists on the userdata partition. The wpa_suppliant.conf file wrote the data to memory as shown in Figure 4.2. This data persists in one instance on both the physical (a) and logical (b) images.

The 2011-NexusOne-1 userdata partition contains 15 instances of the IMSI number. The CheckInService.xml file accounted for 14 of those data instances, while one instance is contained in the account database file.

Count	IMSI
14	915050000000122
1	915050000000122

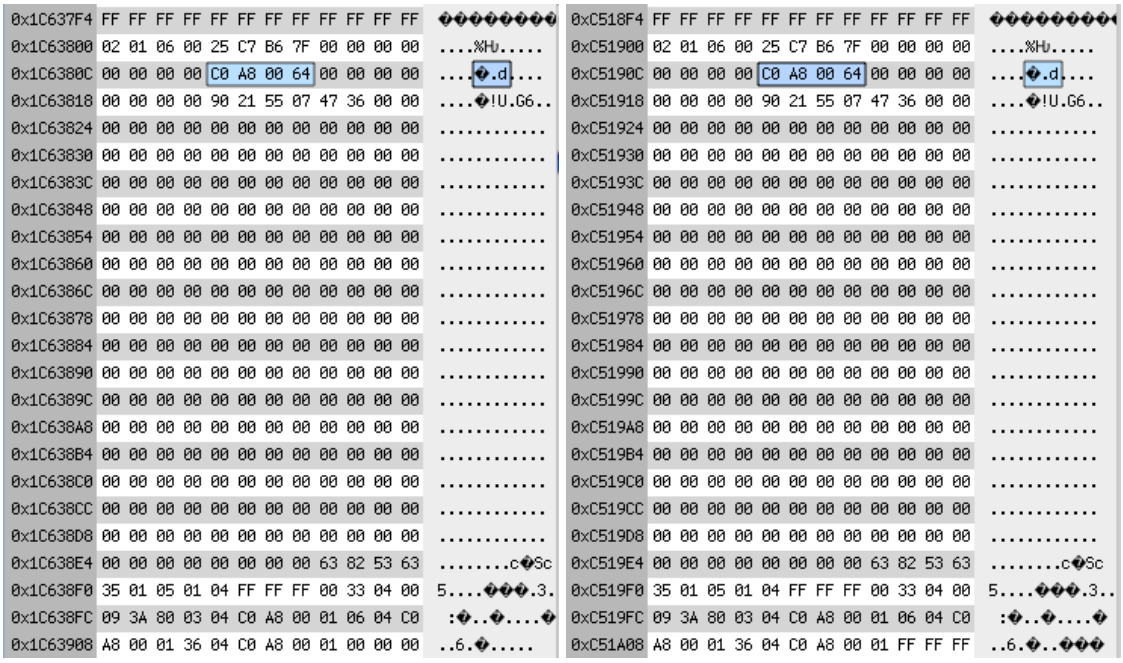
(a) 2011-NexusOne-1

Count	IMSI
1	915050000000122
1	915050000000122

(b) 2011-NexusOne-1-1

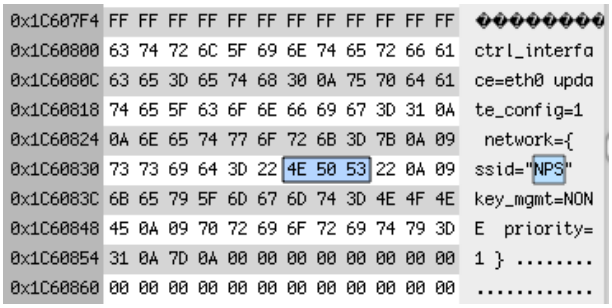
Table 4.7: IMSI captures from the 2011-NexusOne-1/1-1 userdata partitions.

Table 4.8 shows the possible binary IP addresses located in the system partition. The unsigned 32-bit integers represent the IP address of the D-Link router used to access the internet.

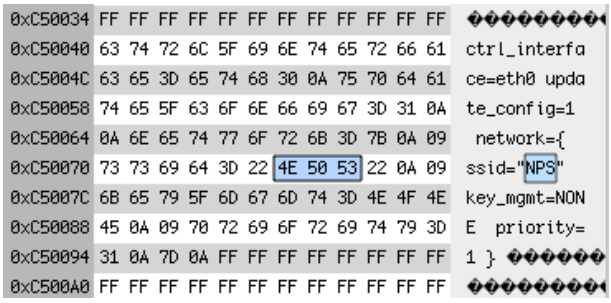


(a) 2011-NexusOne-1 (b) 2011-NexusOne-1-1

Figure 4.1: DHCP ACK found in 2011-NexusOne-1/1-1 `userdata` partitions.



(a) 2011-NexusOne-1



(b) 2011-NexusOne-1-1

Figure 4.2: SSIDs found in 2011-NexusOne-1/1-1 `userdata` partitions.

Count	IP Address	Preceding 2-gram	Partition	Endian
50	192.168.0.1	0x3CC1	mtd3	Little
01	192.168.0.1	0x16EF	mtd3	Little

(a) 2011-NexusOne-1

Count	IP Address	Preceding 2-gram	Partition	Endian
50	192.168.0.1	0x3CC1	system	Little
01	192.168.0.1	0x16EF	system	Little

(b) 2011-NexusOne-1-1

Table 4.8: 2011-NexusOne-1/1-1 Binary IP Address captures.

Nexus S Results:

Item	Count
ASCII IMSI (915050000000120)	9
NPS SSID instance	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
Binary IP address 192.168.0.1 in the system partition	24
Binary IP address 192.168.0.1 in the userdata partition	2
Binary IP address 74.125.224.43 in the radio parttition (mtd5ro).	1

Table 4.9: 2011-NexusS-1 Quick Summary.

Item	Count
ASCII IMSI (915050000000120)	2
NPS SSID instance	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
Binary IP address 192.168.0.1 in the system partition	14
Binary IP address 192.168.0.1 in the userdata partition	2

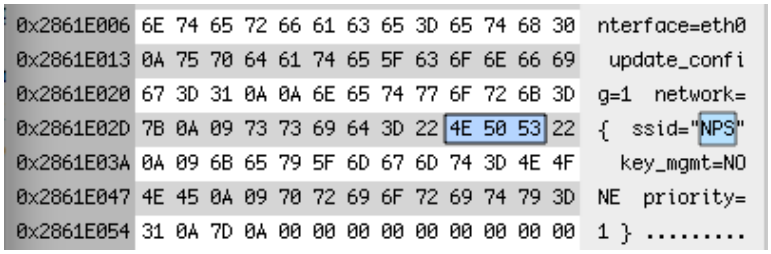
Table 4.10: 2011-NexusS-1-1 Quick Summary.

The Nexus S show behavior very similar to the Nexus One. The required internet access results in the D-Link router's SSID "NPS" persisting on the userdata partition as shown in Figure 4.3. The device also stores DHCP ACKs in the userdata partition as shown in Figure 4.5. The IP address of the Nexus S, 192.168.0.100, is highlighted in Figure 4.5 while the D-Link router's IP address of 192.168.0.1 can be seen in hexadecimal 0xC0A80001 on the bottom of the Figures.

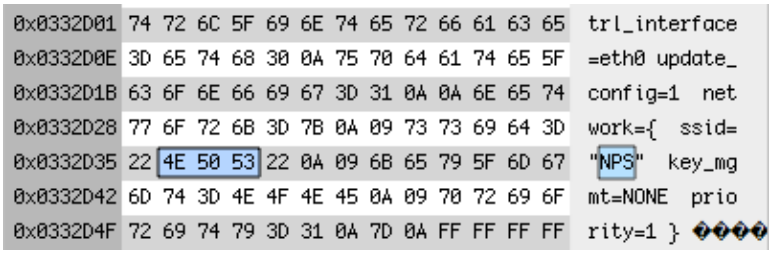
The userdata partition contains 9 instances of the IMSI number itemized in Table 4.11.

As was the case for the Nexus One, the CheckInService.xml file accounts for eight IMSI instances and the /data/ system/ accounts.db file accounts for one instance. The CheckInService.xml

file does have one instance different from the remaining seven. The difference is shown in Figure 4.4, indicated by a "`no-sim`" value prior to the IMSI number. We are unable to determine why the Nexus S `CheckInService.xml` file wrote the `no-sim` value to memory, we leave this to future work.



(a) 2011-NexusS-1



(b) 2011-NexusS-1-1

Figure 4.3: SSIDs found in 2011-NexusS-1/1-1 `userdata` partitions.

Count	IMSI	Preceding 2-gram
7	915050000000120	
1	915050000000120	
1	915050000000120	

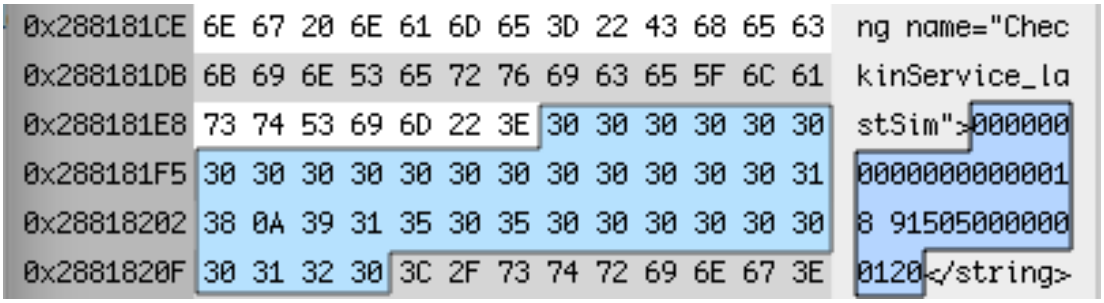
(a) 2011-NexusS-1

Count	IMSI	Preceding 2-gram
1	915050000000120	0x380A
1	915050000000120	0x7369

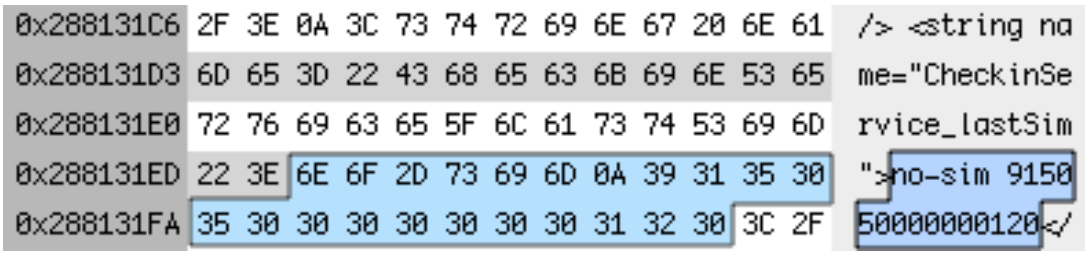
(b) 2011-NexusS-1-1

Table 4.11: IMSI captures from `userdata` partitions.

Table 4.12 shows possible binary IP addresses found in the `system`, `userdata` and `radio` partitions. The `192.168.0.1` IP addresses correlate to the D-Link router's IP address. The `74.125.224.43` IP address found in the radio partition is found in the Wireshark network packet captures taken when the Nexus S connected to the internet and is a Google server address.



(a) Normal.



(b) No-sim instance.

Figure 4.4: IMSIs found in 2011-NexusS-1 `userdata` partitions.

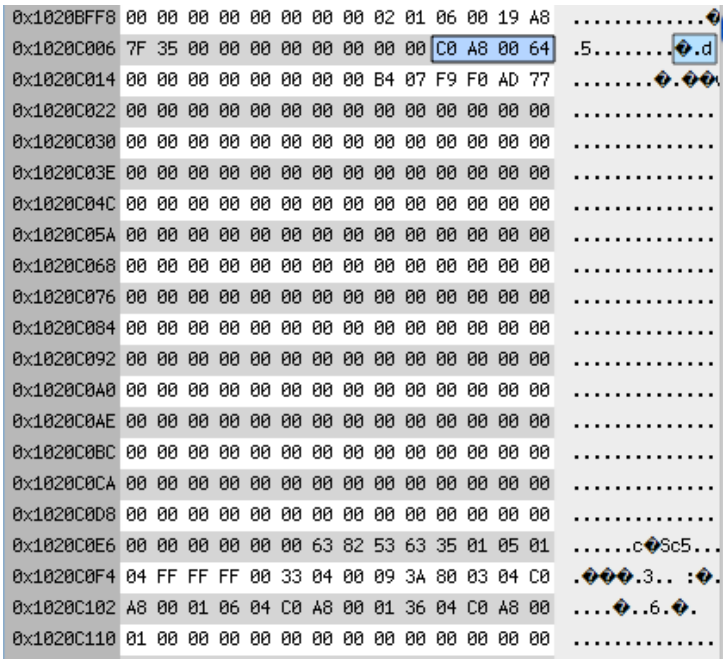
Count	IP Address	Preceding 2-gram	Partition	Endian
01	74.125.224.43	0xCD71	mtd5ro	Little
22	192.168.0.1	0x3CC1	mmcblk0p1	Little
01	192.168.0.1	0xE901	mmcblk0p1	Little
01	192.168.0.1	0x9EC0	mmcblk0p1	Little
01	192.168.0.1	0x38C0	mmcblk0p2	Little
01	192.168.0.1	0xBA55	mmcblk0p2	Little

(a) 2011-NexusS-1

Count	IP Address	Preceding 2-gram	Partition	Endian
12	192.168.0.1	0x3CC1	system	Little
01	192.168.0.1	0xE901	system	Little
01	192.168.0.1	0x9EC0	system	Little
01	192.168.0.1	0x38C0	data	Little
01	192.168.0.1	0xBA55	data	Little

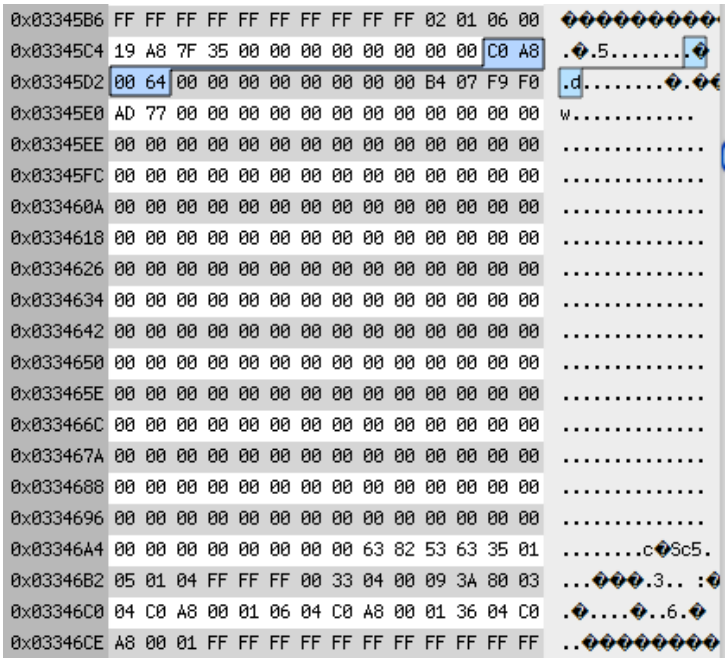
(b) 2011-NexusS-1-1

Table 4.12: 2011-NexusS-1/1-1 Binary IP Address captures.



```
0x1020BFF8 00 00 00 00 00 00 00 00 02 01 06 00 19 A8  ..............
0x1020C006 7F 35 00 00 00 00 00 00 00 00 C0 A8 00 64  .5.........d
0x1020C014 00 00 00 00 00 00 00 00 B4 07 F9 F0 AD 77  .........w
0x1020C022 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C030 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C03E 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C04C 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C05A 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C068 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C076 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C084 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C092 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C0AE 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C0BC 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C0CA 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C0D8 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x1020C0E6 00 00 00 00 00 00 63 82 53 63 35 01 05 01  ......c.Sc5...
0x1020C0F4 04 FF FF FF 00 33 04 00 09 3A 80 03 04 C0  .....3...:....
0x1020C102 A8 00 01 06 04 C0 A8 00 01 36 04 C0 A8 00  .........6....
0x1020C110 01 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
```

(a) 2011-NexusS-1



```
0x03345B6 FF FF FF FF FF FF FF FF FF FF 02 01 06 00  ..............
0x03345C4 19 A8 7F 35 00 00 00 00 00 00 00 00 C0 A8  ...5..........
0x03345D2 00 64 00 00 00 00 00 00 00 00 B4 07 F9 F0  .d............
0x03345E0 AD 77 00 00 00 00 00 00 00 00 00 00 00 00  .w............
0x03345EE 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x03345FC 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x033460A 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x0334618 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x0334626 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x0334634 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x0334642 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x0334650 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x033465E 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x033466C 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x033467A 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x0334688 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x0334696 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
0x03346A4 00 00 00 00 00 00 00 00 63 82 53 63 35 01  ........c.Sc5.
0x03346B2 05 01 04 FF FF FF 00 33 04 00 09 3A 80 03  .......3...:..
0x03346C0 04 C0 A8 00 01 06 04 C0 A8 00 01 36 04 C0  ...........6..
0x03346CE A8 00 01 FF FF FF FF FF FF FF FF FF FF FF  ..............
```

(b) 2011-NexusS-1-1

Figure 4.5: Binary IP addresses found in `userdata` partitions.

4.2.2 Effects of Associating Google Account to Device

The 2011-NexusX-1.5 image associates a Google account to the device to determine what changes occur, if any, to the non-volatile flash memory partitions.

Nexus One Results:

Item	Count
ASCII IMSI (915050000000122)	26
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	2
Binary IP address `192.168.0.1` in the `cache` partition	1

Table 4.13: 2011-NexusOne-1.5 Quick Summary.

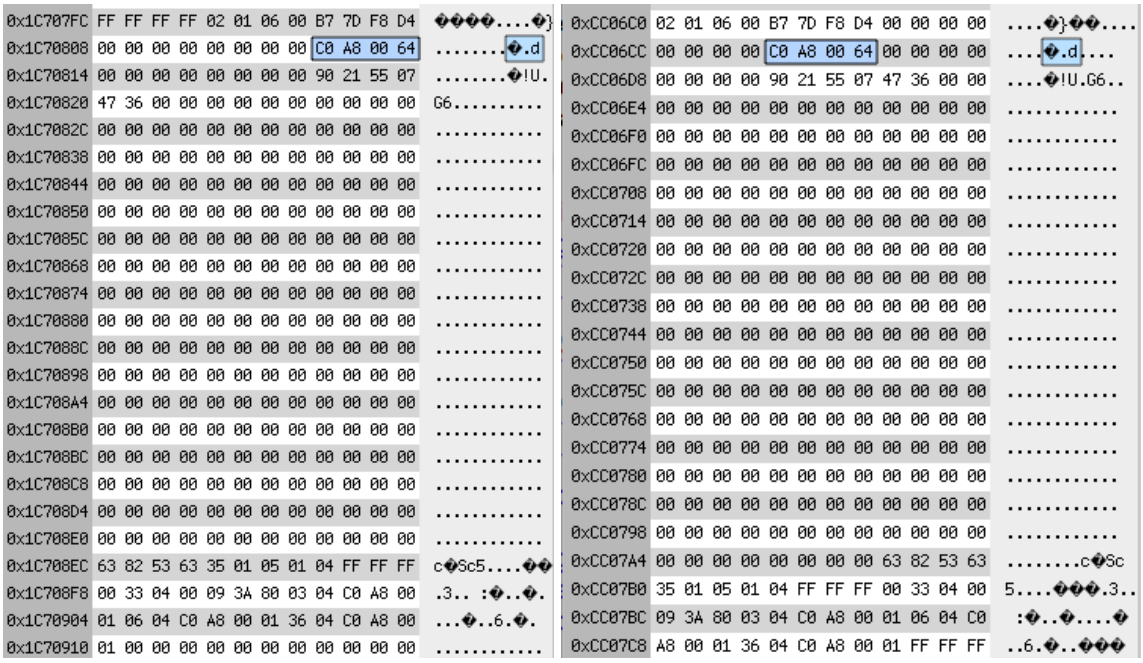
Item	Count
ASCII IMSI (915050000000122)	2
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1
Binary IP address `192.168.0.1` in the `cache` partition	1

Table 4.14: 2011-NexusOne-1.5-1 Quick Summary.

As the quick summary indicates, associating a Google account to an Android device creates network metadata that was found in the baseline image. The following was found to differ from the baseline image.

The 2011-NexusOne-1.5 `userdata` partition contains two instances of the DHCP ACKs while the 2011-NexusOne-1.5-1 `userdata` partition contained one DHCP ACK instance. As shown in Figure 4.6, one instance from the 2011-NexusOne-1.5 image contains the same four-byte Bootstrap Protocol Transaction ID number `0xB77DF8D4` as the 2011-NexusOne-1.5-1 image instance. The second 2011-NexusOne-1.5 image instance contains the Transaction ID `0x071531B9`.

Table 4.15 shows the possible binary IP addresses located in the `cache` partition. The first two unsigned 32-bit integers represent the IP addresses belonging to Google and E-bay. The



(a) 2011-NexusOne-1.5 (b) 2011-NexusOne-1.5-1

Figure 4.6: DHCP ACKs found in `userdata` partitions.

remaining capture represents the D-Link router's IP address.

Count	IP Address	Preceding 2-gram	Partition	Endian
01	74.125.224.101	0x140A	mtd3	Little
01	66.211.171.194	0x91F7	mtd3	Little
01	192.168.0.1	0x4100	mtd4	Little

(a) 2011-NexusOne-1.5

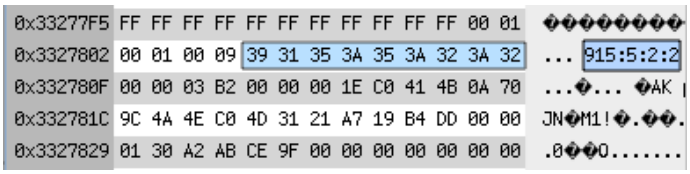
Count	IP Address	Preceding 2-gram	Partition	Endian
01	74.125.224.101	0x140A	system	Little
01	66.211.171.194	0x91F7	system	Little
01	192.168.0.1	0x4100	cache	Little

(b) 2011-NexusOne-1.5-1

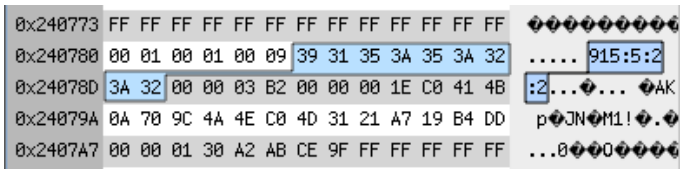
Table 4.15: 2011-NexusOne-1.5/1.5-1 Binary IP Address captures.

Associating the Google account allows analysis of the two Android Location cache files previously mentioned, as empirical tests have shown that the files do not reside on the device until a Google account is associated to the phone. The `cache.cell` file wrote to disk in the 2011-NexusOne-7 and 2011-NexusOne-7-1 experiments as shown in Figure 4.7. The `cache.wifi` file wrote to disk in the 2011-NexusOne-7 and 2011-NexusOne-7-1 experiments as shown in

Figure 4.8. These files are valuable in that they show cellular base station and wireless router metadata coded in ASCII.

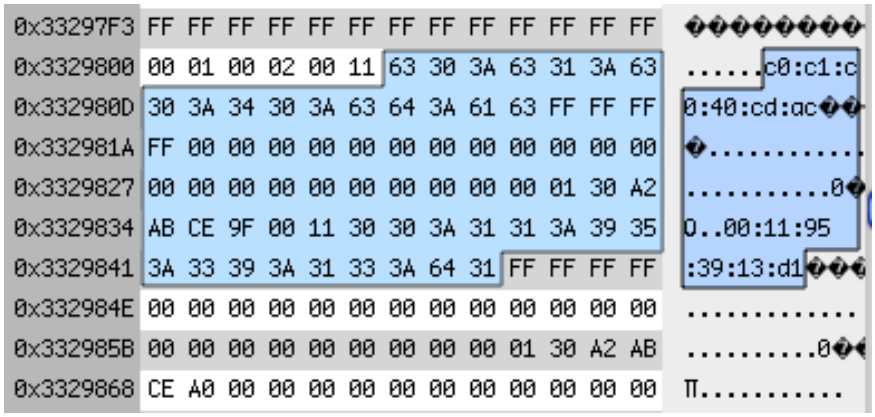


(a) 2011-NexusOne-7

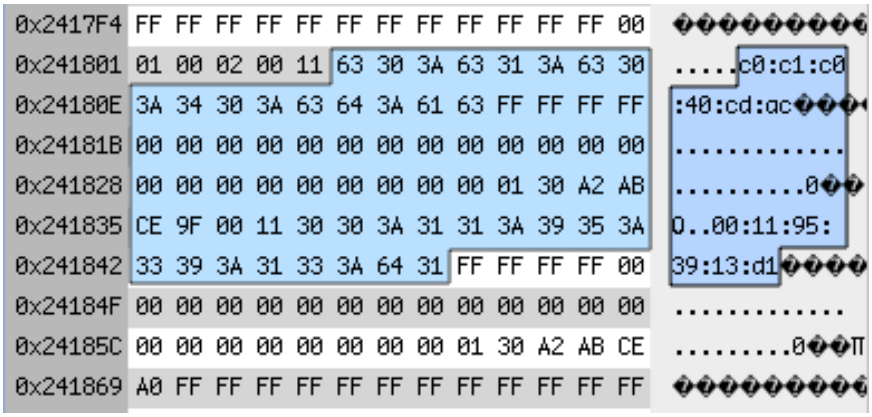


(b) 2011-NexusOne-7-1

Figure 4.7: CIDs found in 2011-NexusOne-7/7-1 user data partitions.



(a) 2011-NexusOne-7



(b) 2011-NexusOne-7-1

Figure 4.8: Router MAC Addresses found in 2011-NexusOne-7/7-1 `userdata` partitions.

Nexus S Results:

Item	Count
ASCII IMSI (915050000000120)	10
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1
Binary IP address `192.168.0.1` in the `data` partition	2
Binary IP address `66.211.171.194` in the `system` partition	2

Table 4.16: 2011-NexusS-1.5 Quick Summary.

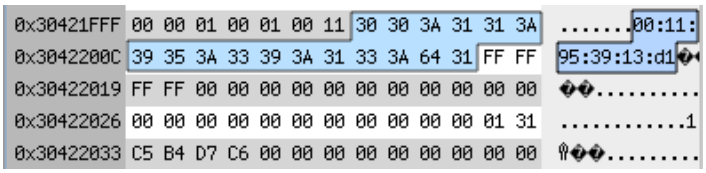
Item	Count
ASCII IMSI (915050000000120)	2
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1
Binary IP address `192.168.0.1` in the `data` partition	2
Binary IP address `66.211.171.194` in the `system` partition	1

Table 4.17: 2011-NexusS-1.5-1 Quick Summary.

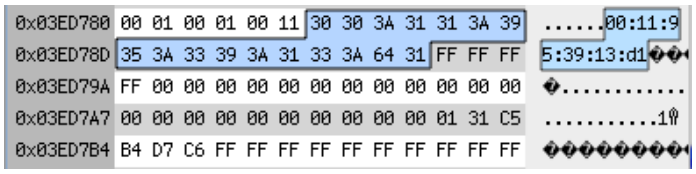
As the quick summary indicates, associating a Google account to an Android device creates network metadata that was found in the baseline image. The following was found to differ from the baseline image.

Associating a Google account to the Nexus S provided similar results that were found in the Nexus One. The two Android Location package files, cache.wifi and cache.cell write cellular base station and wireless router metadata coded in ASCII to the `userdata` partition. The Nexus S differs in that it wrote the cache.wifi file in the 2011-NexusS-1.5 image whereas the Nexus One did not. Figure 4.9 shows the 2011-NexusS-1.5 and 2011-NexusS-1.5-1 captures of the D-Link's MAC address written by the cache.wifi file. Figures 4.10 and 4.11 show these file's activities in the 2011-NexusS-7 and 2011-NexusS7-1 images. With the TacBSR present in those experiments, the cache.cell file writes the TacBSR's metadata as shown in Figure 4.10.

Figure 4.11 shows the cache.wifi file's activities writing the D-Link and Cisco routers MAC addresses.

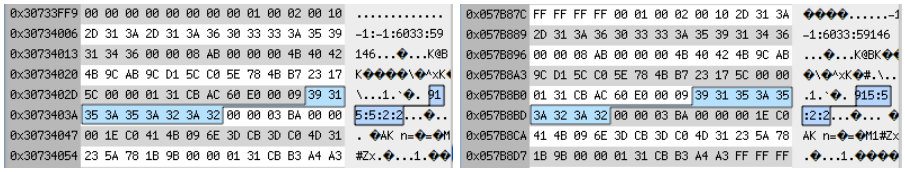


(a) 2011-NexusS-1.5



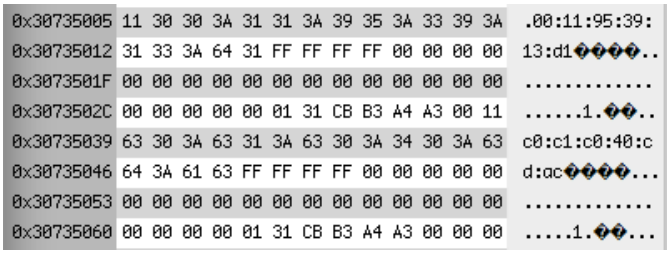
(b) 2011-NexusS-1.5-1

Figure 4.9: D-Link MAC addresses found in `userdata` partitions.

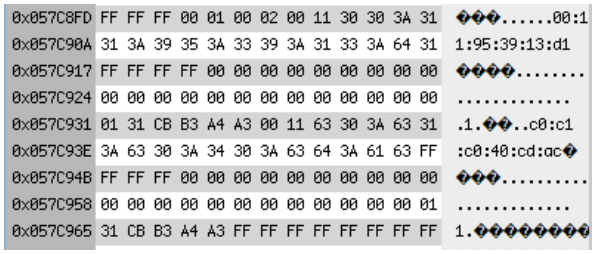


(a) 2011-NexusS-7 (b) 2011-NexusS-7-1

Figure 4.10: CIDs found in 2011/NexusS-7/7-1 `userdata` partitions.



(a) 2011-NexusS-7



(b) 2011-NexusS-7-1

Figure 4.11: Router MAC Addresses found in 2011-NexusS-7/7-1 `userdata` partitions.

Table 4.18 shows the possible binary IP addresses located in different partitions. `192.168.0.1` represents the IP address of the D-Link router used to access the internet. The `66.211.171.194` IP address was indicated in the Wireshark network capture and correlates to E-bay address range.

Count	IP Address	Preceding 2-gram	Partition	Endian
02	66.211.171.194	0x91F7	mmcblk0p1	Little
01	192.168.0.1	0x38C0	mmcblk0p2	Little
01	192.168.0.1	0xBA55	mmcblk0p2	Little

(a) 2011-NexusS-1.5

Count	IP Address	Preceding 2-gram	Partition	Endian
01	66.211.171.194	0x91F7	system	Little
01	192.168.0.1	0x38C0	data	Little
01	192.168.0.1	0xBA55	data	Little

(b) 2011-NexusS-1.5-1

Table 4.18: 2011-NexusS-1.5/1.5-1 Binary IP Address captures.

4.2.3 Effects of File Transfer via Bluetooth

The 2011-NexusX-3 image transfers three files of different sizes across a Bluetooth connection to determine if any Bluetooth-specific metadata persists on the non-volatile flash memory.

Nexus One Results:

The Bluetooth experiment yielded similar results to the 2011-NexusOne-1.5 experiment, due to accessing the internet to flash a `ROM Manager` recovery ROM.

Item	Count
ASCII IMSI (915050000000122)	23
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1

Table 4.19: 2011-NexusOne-3 Quick Summary.

Item	Count
ASCII IMSI (915050000000122)	2
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1

Table 4.20: 2011-NexusOne-3-1 Quick Summary.

The 2011-NexusOne-3 `userdata` dump contains multiple ASCII data that correlates to the Macbook Pro's Bluetooth MAC address (BTADDR). A total of 1513 instances of the Macbook Pro's BTADDR address were retrieved. The total captured addresses, itemized by their two preceding bytes, are shown in Table 4.21.

The file responsible for the most BTADDR instances is the `btopp.db` database, along with its associated `btopp.db-journal` file. The Bluetooth Object Push Profile database accounted for 1479 BTADDR instances. Those instances are the cumulative count from the `0x6E01`, `0x6701` and `0x6601` Preceding 2-grams shown in Table 4.21. Those three Preceding 2-grams relate to the `to_build_be.txt`, `androidarch.jpg` and `trailheadmap.pdf` files sent from the Macbook Pro and saved by the btopp.db file.

Other instances of discovered BTADDRs were within various other files totaling 34 occurrences over 12 files. The files writing Bluetooth metadata to memory can be found in Table 4.22.

BTADDR instances from the logical dump decline relative to the 2011-NexusOne-3 dump. The total captures totaled 22. The Preceding 2-gram breakdown is shown in Table 4.21.

BTADDR	Preceding 2-gram	Count
00:26:08:BC:D4:14	0x6E01	544
00:26:08:BC:D4:14	0x6701	517
00:26:08:BC:D4:14	0x6601	418
00:26:08:BC:D4:14	0xFFFF	15
00:26:08:BC:D4:14	0x795F	13
00:26:08:BC:D4:14	0x390A	3
00:26:08:BC:D4:14	0x300A	2
00:26:08:BC:D4:14	0x330A	1

(a) 2011-NexusOne-3

BTADDR	Preceding 2-gram	Count
00:26:08:BC:D4:14	0x6E01	1
00:26:08:BC:D4:14	0x6701	1
00:26:08:BC:D4:14	0x6601	1
00:26:08:BC:D4:14	0xFFFF	10
00:26:08:BC:D4:14	0x795F	6
00:26:08:BC:D4:14	0x390A	1
00:26:08:BC:D4:14	0x300A	1
00:26:08:BC:D4:14	0x330A	1

(b) 2011-NexusOne-3-1

Table 4.21: Bluetooth MAC captures found in `userdata` partitions.

Preceding-2gram	File
0x6E01	btopp.db and btopp.db-journal
0x6701	btopp.db and btopp.db-journal
0x6601	btopp.db and btopp.db-journal
0x795F	settings.db and settings.db-journal
0x390A	sdp
0x300A	sdp
0x330A	sdp
0x795F	classes, eir, names, lastseen, lastused, features, manufacturers, linkkeys, sdp, profiles

Table 4.22: 2011-NexusOne-3 files writing Bluetooth metadata by 2-gram.

Nexus S Results:

Item	Count
ASCII IMSI (915050000000120)	6
NPS SSID instance	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
D-Link router MAC Address 00:11:95:39:13:d1	1

Table 4.23: 2011-NexusS-3 Quick Summary.

Item	Count
ASCII IMSI (915050000000120)	2
NPS SSID instance	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
D-Link router MAC Address 00:11:95:39:13:d1	1

Table 4.24: 2011-NexusS-3-1 Quick Summary.

The Nexus S BTADDR instances show similar results to those found in the Nexus One as indicated in Table 4.25. When the mmcblk0p2 image is mounted, it provides greater detail on the files writing to disk and their contents. The /data/ data/ com.android.bluetooth/ databases/ btopp.db file is responsible for most of the BTADDR instances found as was the case for the Nexus One. The top three preceding 2-grams, 0x6E01, 0x6701 and 0x6601 relate to to_build_be.txt, androidarch.jpg and trailheadmap.pdf files respectively. Under the directory /data/ misc/ bluetoothd/ 78:47:1D:B5:F1:06/ the Nexus S Bluetooth daemon contains the files responsible for the BTADDRs found in the userdata partition, as shown in Table 4.26.

BTADDR	Preceding 2-gram	Count
00:26:08:BC:D4:14	0x6E01	182
00:26:08:BC:D4:14	0x6701	173
00:26:08:BC:D4:14	0x6601	140
00:26:08:BC:D4:14	0xFFFF	10
00:26:08:BC:D4:14	0x380A	2
00:26:08:BC:D4:14	0x795F	9
00:26:08:BC:D4:14	0x390A	1
00:26:08:BC:D4:14	0x300A	1
00:26:08:BC:D4:14	0x330A	1

(a) 2011-NexusS-3

BTADDR	Preceding 2-gram	Count
00:26:08:BC:D4:14	0x6E01	1
00:26:08:BC:D4:14	0x6701	1
00:26:08:BC:D4:14	0x6601	1
00:26:08:BC:D4:14	0xFFFF	8
00:26:08:BC:D4:14	0x380A	2
00:26:08:BC:D4:14	0x795F	6
00:26:08:BC:D4:14	0x390A	1
00:26:08:BC:D4:14	0x300A	1
00:26:08:BC:D4:14	0x330A	1

(b) 2011-NexusS-3-1

Table 4.25: Bluetooth MAC captures found in `userdata` partitions.

Preceding-2gram	File
0x6E01	btopp.db and btopp.db-journal
0x6701	btopp.db and btopp.db-journal
0x6601	btopp.db and btopp.db-journal
0x795F	settings.db and settings.db-journal
0x390A	sdp
0x300A	sdp
0x330A	sdp
0x380A	sdp
0x795F	classes, names, lastused, features, manufacturers, linkkeys, sdp, profiles

Table 4.26: 2011-NexusS-3 files writing Bluetooth metadata by 2-gram.

4.2.4 Effects of File Transfers

The 2011-NexusX-4 and 2011-NexusX-5 images transfer one file from a web server to the Android device to determine whether any network metadata persists on the non-volatile flash memory. One wireless router provides access to the external internet, while another wireless router provides access to the internal web server.

Nexus One Results:

Item	Count
ASCII IMSI (915050000000122)	26
`NPS` SSID instance	2
`AndroidForensics` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1
DHCP ACK instance containing binary IP Addresses `192.168.1.1` and `192.168.1.146`	1
Binary IP address `192.168.1.1` in the `system` partition	16
Binary IP address `192.168.1.1` in the `data` partition	3

Table 4.27: 2011-NexusOne-4 Quick Summary.

Item	Count
ASCII IMSI (915050000000122)	2
`NPS` SSID instance	1
`AndroidForensics` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.1.1` and `192.168.1.146`	1
Binary IP address `192.168.1.1` in the `system` partition	16
Binary IP address `192.168.1.1` in the `data` partition	3

Table 4.28: 2011-NexusOne-4-1 Quick Summary.

The file transfers are the first case where both wireless routers are used, providing a unique look into association behaviors. First, the `wpa_supplicant.conf` file contents were found twice

Item	Count
ASCII IMSI (915050000000122)	26
NPS SSID instance	2
AndroidForensics SSID instance	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
DHCP ACK instance containing binary IP Addresses 192.168.1.1 and 192.168.1.144	1
Binary IP address 192.168.1.1 in the data partition	3

Table 4.29: 2011-NexusOne-5 Quick Summary.

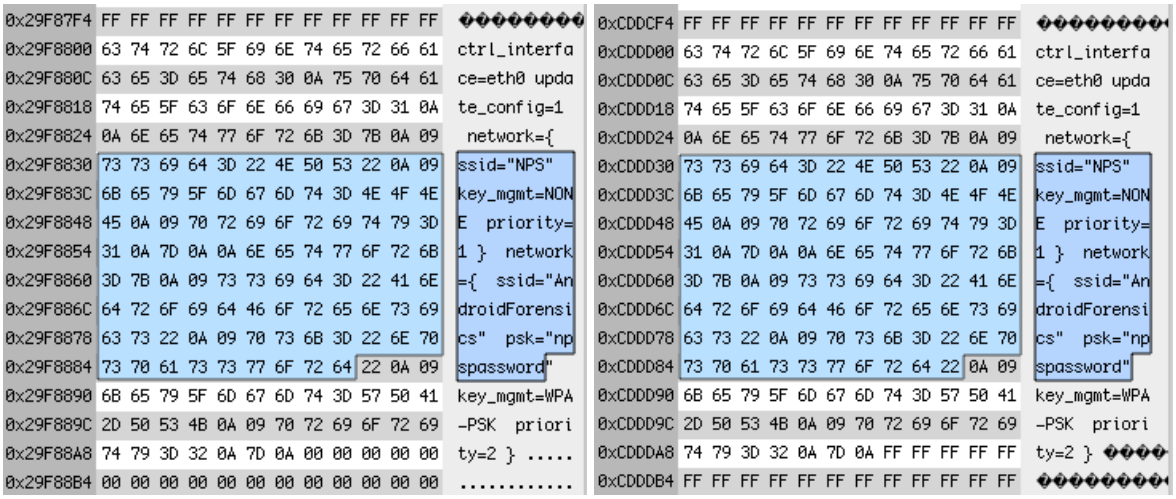
Item	Count
ASCII IMSI (915050000000122)	2
NPS SSID instance	1
AndroidForensics SSID instance	1
DHCP ACK instance containing binary IP Addresses 192.168.1.1 and 192.168.1.144	1
Binary IP address 192.168.1.1 in the data partition	3

Table 4.30: 2011-NexusOne-5-1 Quick Summary.

in the 2011-NexusOne-4 userdata partition The first instance contained the association to the D-Link router with the "NPS" SSID. The second instance had the D-Link router's "NPS" SSID as well as the Cisco router's "AndroidForensics" SSID and its related password as shown in 4.12 (a). As shown in 4.12 (b), the 2011-NexusOne-4-1 userdata partition contains one instance of the wpa_supplicant.conf file whose contents include both router's SSIDs.

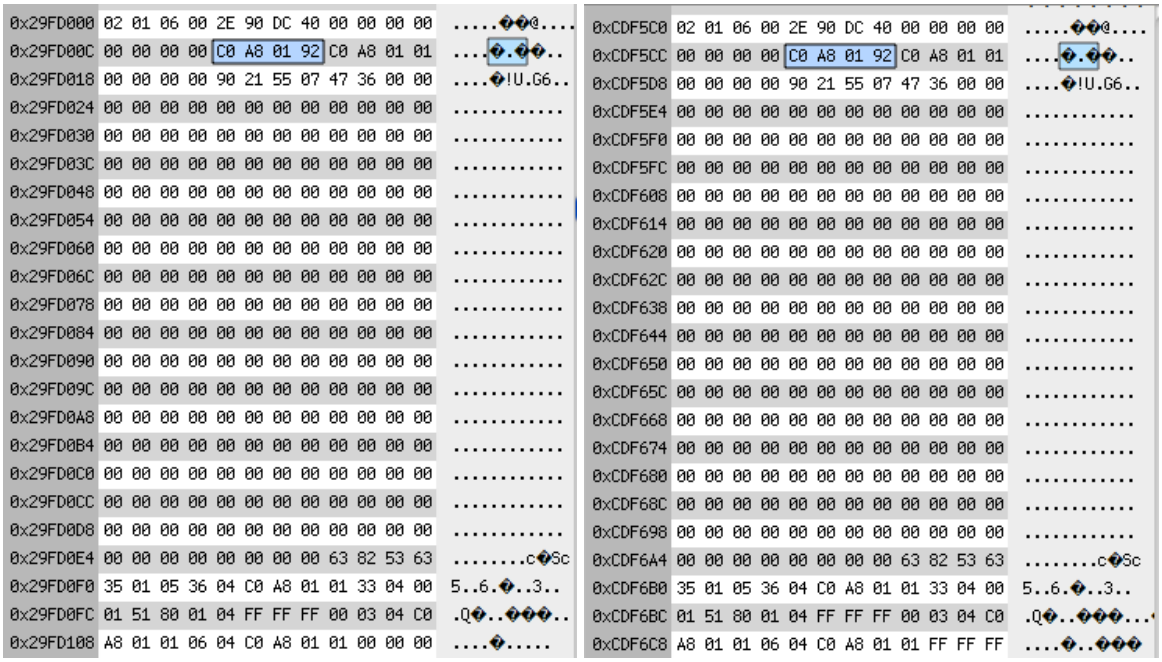
Second, the dhcp-eth0.lease file contents were found in the 2011-NexusOne-4 userdata partition as found in the baseline image, and another instance was found in the same partition for the association to the web server router as shown in Figure 4.13 (a). The 2011-NexusOne-4-1 userdata partition retained one instance of the file as shown in Figure 4.13 (b). The binary IP address found in both images has the value of 192.168.1.146 and relates to the IP address given to the Android device.

The 2011-NexusOne-5 and 2011-NexusOne-5-1 images show the same behavior as indicated in the quick summary.



(a) 2011-NexusOne-4 (b) 2011-NexusOne-4-1

Figure 4.12: wpa_supplicant.conf found in `userdata` partitions.



(a) 2011-NexusOne-4 (b) 2011-NexusOne-4

Figure 4.13: DHCP ACKs found in `userdata` partitions.

Table 4.31 shows the possible binary IP addresses located in the `system` and `userdata` partitions. The unsigned 32-bit integers represent the IP address of the Cisco router used to access

the web server. All possible binary IP address instances are present in the 2011-NexusOne-4, 2011-NexusOne-4-1, 2011-NexusOne-5 and 2011-NexusOne-5-1 images so they are presented in one table.

Count	IP Address	Preceding 2-gram	Partition	Endian
01	192.168.1.1	0x04AC	mtd3	Big
01	192.168.1.1	0x8101	mtd3	Big
01	192.168.1.1	0x9B01	mtd3	Little
01	192.168.1.1	0xBF01	mtd3	Little
01	192.168.1.1	0xCF01	mtd3	Little
01	192.168.1.1	0xC050	mtd3	Little
01	192.168.1.1	0xBF53	mtd3	Little
01	192.168.1.1	0x3863	mtd3	Little
01	192.168.1.1	0xC06B	mtd3	Little
01	192.168.1.1	0x8601	mtd3	Little
01	192.168.1.1	0x8C01	mtd3	Little
01	192.168.1.1	0xA301	mtd3	Little
01	192.168.1.1	0xD6CB	mtd3	Little
01	192.168.1.1	0xBF13	mtd3	Little
01	192.168.1.1	0xC00E	mtd3	Little
01	192.168.1.1	0xC008	mtd3	Little
01	192.168.1.1	0xBF3F	mtd5	Little
01	192.168.1.1	0xC021	mtd5	Little
01	192.168.1.1	0xC007	mtd5	Little

Table 4.31: Binary IP Address captures found in all Nexus One file transfer experiments.

Nexus S Results:

Item	Count
ASCII IMSI (915050000000120)	8
NPS SSID instance	1
AndroidForensics SSID instance	1
D-Link router MAC Address 00:11:95:39:13:d1	1
DHCP ACK instance containing binary IP Addresses 192.168.1.1 and 192.168.1.147	1
Binary IP address 192.168.1.1 in the system partition	10
Binary IP address 192.168.1.1 in the userdata partition	7

Table 4.32: 2011-NexusS-4 Quick Summary.

Item	Count
ASCII IMSI (915050000000120)	2
NPS SSID instance	1
AndroidForensics SSID instance	1
D-Link router MAC Address 00:11:95:39:13:d1	1
DHCP ACK instance containing binary IP Addresses 192.168.1.1 and 192.168.1.147	1
Binary IP address 192.168.1.1 in the userdata partition	7

Table 4.33: 2011-NexusS-4-1 Quick Summary.

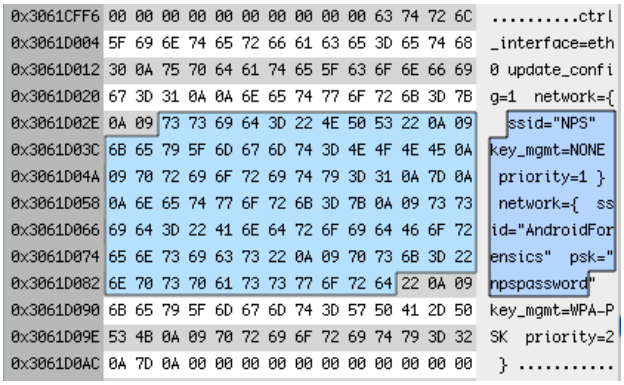
The Nexus S wpa_supplicant.conf file contents were found in the userdata partitions as shown in Figure 4.14. The Nexus S stored the D-Link and Cisco router's SSIDs, but does not have an instance with only the initial D-Link router's SSID, NPS, as was the case with Nexus One userdata partitions. The DHCP ACKs exhibit the same behavior as seen on the Nexus One. The 2011-NexusS-4 userdata partition holds the DHCP ACK from the web server router, but does not hold the DHCP ACK from the D-Link router to the internet. The 2011-NexusS-5 and 2011-NexusS-5-1 images contain similar data when compared to the 2011-NexusS-4 images. The 2011-NexusS-5 DHCP ACKs are shown in Figure 4.15 with their 192.168.1.144 IP address highlighted.

Item	Count
ASCII IMSI (915050000000120)	8
NPS SSID instance	1
AndroidForensics SSID instance	1
D-Link router MAC Address 00:11:95:39:13:d1	1
DHCP ACK instance containing binary IP Addresses 192.168.1.1 and 192.168.1.144	1
Binary IP address 192.168.1.1 in the userdata partition	7
Binary IP address 149.20.68.17 in the userdata partition	2

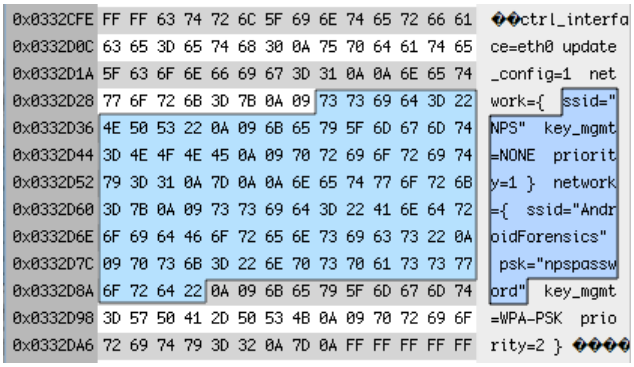
Table 4.34: 2011-NexusS-5 Quick Summary.

Item	Count
ASCII IMSI (915050000000120)	2
NPS SSID instance	1
AndroidForensics SSID instance	1
D-Link router MAC Address 00:11:95:39:13:d1	1
DHCP ACK instance containing binary IP Addresses 192.168.1.1 and 192.168.1.144	1
Binary IP address 192.168.1.1 in the userdata partition	7
Binary IP address 149.20.68.17 in the userdata partition	1

Table 4.35: 2011-NexusS-5-1 Quick Summary.

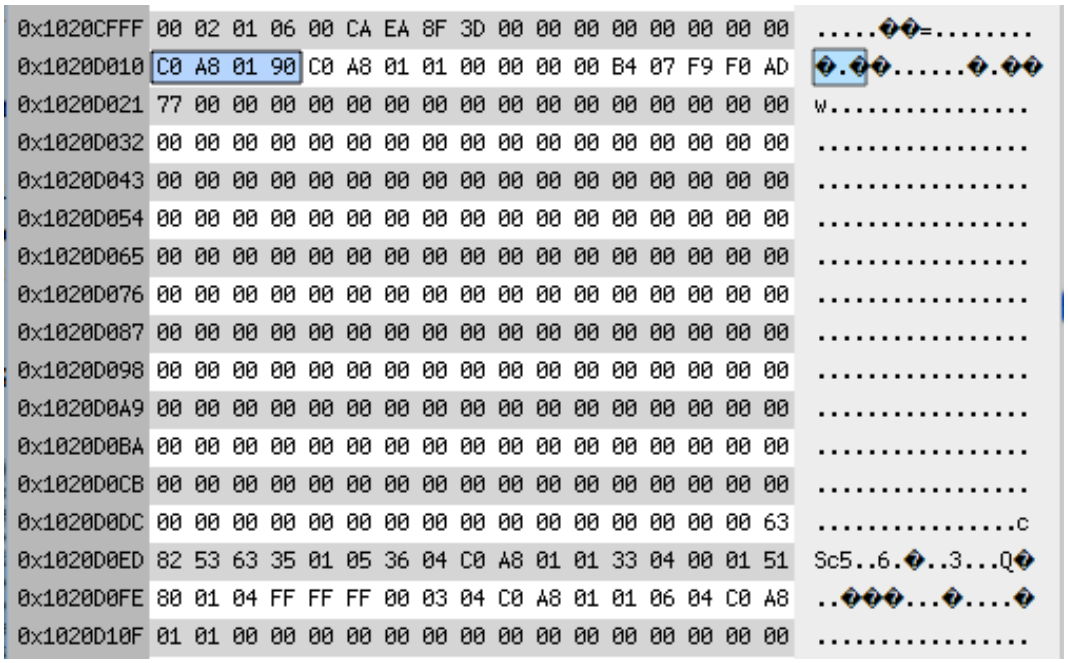


(a) 2011-NexusS-4

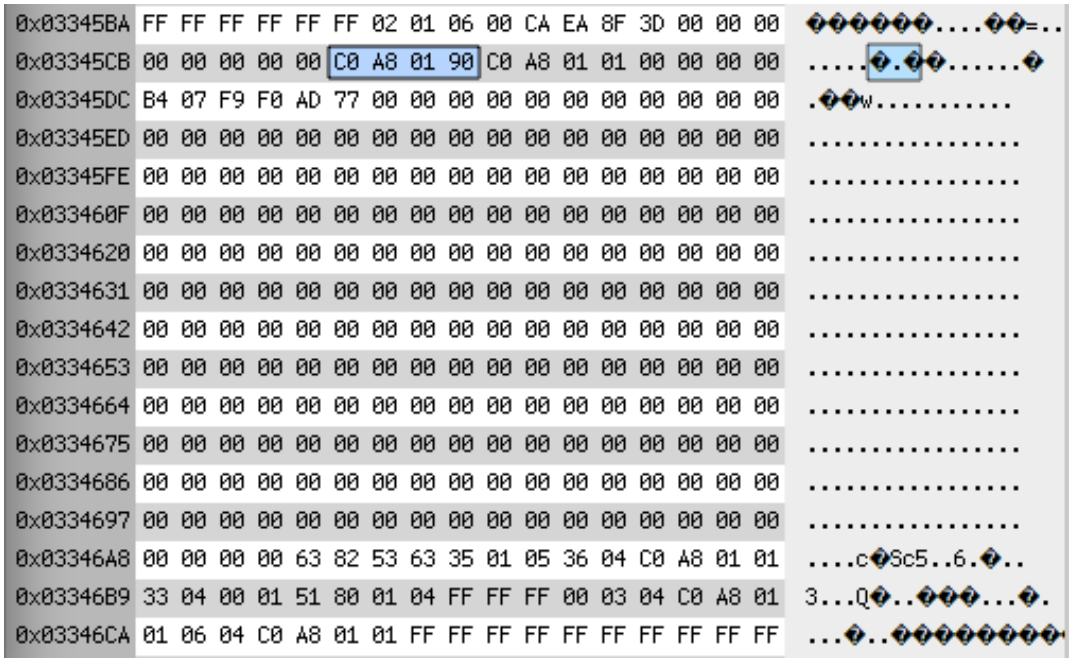


(b) 2011-NexusS-4-1

Figure 4.14: wpa_supplicant.conf found in `userdata` partitions.



(a) 2011-NexusS-5



(b) 2011-NexusS-5-1

Figure 4.15: DHCP ACKs found in `userdata` partitions.

Count	IP Address	Preceding 2-gram	Partition	Endian
01	192.168.1.1	0xC07C	mmcblk0p1	Little
01	192.168.1.1	0xBF21	mmcblk0p1	Little
01	192.168.1.1	0xC033	mmcblk0p1	Little
02	192.168.1.1	0xC009	mmcblk0p1	Little
02	192.168.1.1	0xC00A	mmcblk0p1	Little
01	192.168.1.1	0xBF53	mmcblk0p1	Little
01	192.168.1.1	0xC07C	mmcblk0p1	Little
01	192.168.1.1	0xBF7E	mmcblk0p1	Little
01	192.168.1.1	0xF201	mmcblk0p1	Little
01	192.168.1.1	0xBF09	mmcblk0p1	Little
01	192.168.0.1	0x38C0	mmcblk0p2	Little
01	192.168.0.1	0xBA55	mmcblk0p2	Little
01	192.168.1.1	0xC002	mmcblk0p2	Little
01	192.168.1.1	0xBF31	mmcblk0p2	Little
01	192.168.1.1	0x9F01	mmcblk0p2	Little
01	192.168.1.1	0xEE01	mmcblk0p2	Little
01	192.168.1.1	0xBF09	mmcblk0p2	Little
01	192.168.1.1	0xF6A1	mmcblk0p2	Little
01	192.168.1.1	0x8001	mmcblk0p2	big

(a) 2011-NexusS-4

Count	IP Address	Preceding 2-gram	Partition	Endian
01	192.168.0.1	0x38C0	data	Little
01	192.168.0.1	0xBA55	data	Little
01	192.168.1.1	0xC002	data	Little
01	192.168.1.1	0xBF31	data	Little
01	192.168.1.1	0x9F01	data	Little
01	192.168.1.1	0xEE01	data	Little
01	192.168.1.1	0xBF09	data	Little
01	192.168.1.1	0xF6A1	data	Little
01	192.168.1.1	0x8001	data	Big

(b) 2011-NexusS-4-1

Table 4.36: 2011-NexusS-4/4-1 Binary IP Address captures.

Count	IP Address	Preceding 2-gram	Partition	Endian
02	149.20.68.17	0x4D11	system	Little
01	192.168.0.1	0x38C0	mmcblk0p2	Little
01	192.168.0.1	0xBA55	mmcblk0p2	Little
01	192.168.1.1	0xC002	mmcblk0p2	Little
01	192.168.1.1	0xBF31	mmcblk0p2	Little
01	192.168.1.1	0x9F01	mmcblk0p2	Little
01	192.168.1.1	0xEE01	mmcblk0p2	Little
01	192.168.1.1	0xBF09	mmcblk0p2	Little
01	192.168.1.1	0xF6A1	mmcblk0p2	Little
01	192.168.1.1	0x8001	mmcblk0p2	big

(a) 2011-NexusS-5

Count	IP Address	Preceding 2-gram	Partition	Endian
01	149.20.68.17	0x4D11	system	Little
01	192.168.1.1	0xF6A1	data	Little
01	192.168.1.1	0x8001	data	Big

(b) 2011-NexusS-5-1

Table 4.37: 2011-NexusS-5/5-1 Binary IP Address captures.

4.2.5 Effects of Application Use

The 2011-NexusX-6 image is created after using the `Facebook` application to determine whether the application will store relevant network metadata to the non-volatile flash memory.

Nexus One Results:

Item	Count
ASCII IMSI (915050000000122)	24
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	2
Binary IP address `131.188.3.220` in the `data` partition	1

Table 4.38: 2011-NexusOne-6 Quick Summary.

Item	Count
ASCII IMSI (915050000000122)	2
`NPS` SSID instance	1
DHCP ACK instance containing binary IP Addresses `192.168.0.1` and `192.168.0.100`	1
Binary IP address `131.188.3.220` in the `data` partition	1

Table 4.39: 2011-NexusOne-6-1 Quick Summary.

Table 4.40 shows the possible binary IP addresses located in the `cache` partition. The discovered IP address corresponds to a Network Time Protocol (NTP) server and similarly occurs in the pcap file for the experiment. NTP is a protocol designed to synchronize computer clocks over a network. Review of the Wireshark pcap file shows no indication of the 0x74EC value that precedes the possible IP address. This indicates that the binary IP found is not in a IP packet structure.

Count	IP Address	Preceding 2-gram	Partition	Endian
01	131.188.3.220	0x74EC	mtd4	Little

(a) 2011-NexusOne-6

Count	IP Address	Preceding 2-gram	Partition	Endian
01	131.188.3.220	0x74EC	cache	Little

(b) 2011-NexusOne-6-1

Table 4.40: 2011-NexusOne-6/6-1 Binary IP Address captures.

Nexus S Results:

Item	Count
ASCII IMSI (915050000000120)	2
NPS SSID instance	1
D-Link router MAC Address 00:11:95:39:13:d1	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
Binary IP address 130.149.17.8 in the userdata partition	1

Table 4.41: 2011-NexusS-6 Quick Summary.

Item	Count
ASCII IMSI (915050000000120)	1
NPS SSID instance	1
D-Link router MAC Address 00:11:95:39:13:d1	1
DHCP ACK instance containing binary IP Addresses 192.168.0.1 and 192.168.0.100	1
Binary IP address 130.149.17.8 in the userdata partition	1

Table 4.42: 2011-NexusS-6-1 Quick Summary.

Table 4.43 shows a possible binary IP address retrieved from the system partition. The address correlates to a range of addresses that belong to NTP servers similar to the Nexus One capture.

Count	IP Address	Preceding 2-gram	Partition	Endian
01	130.149.17.8	0x0000	system	Little

(a) 2011-NexusS-6

Count	IP Address	Preceding 2-gram	Partition	Endian
01	130.149.17.8	0x0000	system	Little

(b) 2011-NexusS-6-1

Table 4.43: 2011-NexusS-6/6-1Binary IP Address captures.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions and Future Work

5.1 Conclusions

This thesis asked one primary question: can network artifacts be retrieved from Android mobile Smartphones to identify the device's previous network access points. To answer this question, a controlled environment was developed to transfer data across network nodes. Wireless routers were used to transfer data from a web server to the Android device and another wireless router ultimately provided access to the larger internet. An external device was paired to the Android devices via Bluetooth and conducted file transfers. The phones associated with a cellular base station to determine if any cellular network metadata persists on the phone's internal non-volatile memory.

We investigated this thesis with two Android Smartphones; a HTC manufactured Nexus One and a Samsung manufactured Nexus S. The two phones were chosen for their different files systems currently used by the Android operating system. The Nexus One has a YAFFS2 flash file system while the Nexus S contains an EXT4 file system. For acquisition, each device's internal non-volatile memory was retrieved using the Android SDK `adb` shell to pull the data to an examiner's computer. Through the use of file carving tools and a hex editor, we discovered the following residual network data structures:

Nexus One:

- Allocated and non-allocated pages containing wireless 802.11 router Service Set Identifiers (SSID) recovered from the `userdata` partition. SSIDs are ASCII coded string identifiers that are provided to wireless access points. The `wpa_supplicant.conf` file wrote to disk the router's SSID and associated password when the Android device associated to the router.

- Allocated and non-allocated pages containing wireless router International Mobile Subscriber Identity (IMSI) from the `userdata` partition. IMSIs are unique and can be attributed to a specific user. Two files consistently provided IMSI data: the `CheckinService.xml` and `accounts.db` files. The `CheckinService.xml` file is an Android service that searches for updates to downloaded Android applications and Android OS firmware.

- Allocated and non-allocated pages containing 802.11 access point DHCP ACKs from the `userdata` partition. DHCP ACKs contain the unsigned 32-bit IPv4 address of the Android Device and the wireless router providing network services.
- Allocated pages containing cellular base station metadata, including the Mobile Network Code (MNC), Mobile Country Code (MCC), Local Area Code (LAC) and Cell Identification (CID) from the `userdata` partition. These four base station metadata can be used to physically locate a cellular tower. The Android location package file `cache.cell` located under the `/data/data/com.google.android.location/files` directory provides the retrieved metadata.
- The `userdata` partition contains allocated pages containing wireless router Media Access Control (MAC) addresses coded in ASCII. The Android location package file `cache.wifi` located under the `/data/data/com.google.android.location/files` directory provides the retrieved metadata.
- The `userdata` partition contains allocated and non-allocated pages containing Bluetooth MAC addresses of devices paired with the phone. Multiple files store the Bluetooth address of paired devices, including the Bluetooth daemon and its related files and the `btopp.db` file.

Nexus S:
The data acquired from the Nexus S was identical from the Nexus One with the exception when two wireless routers were used in an experiment. The current associated router's DHCP ACK was found in non-volatile memory. The allocated `wpa_supplicant.conf` file was retrieved containing both routers SSIDs. We were not able to retrieve a DHCP ACK from the first associated router and an unallocated `wpa_supplicant.conf` file as found in the Nexus One.

This thesis contributed the following to the field of mobile forensics:

- The first known Android Smartphone corpus with delineated network metadata.
- A thorough review of network data structures that reside in Android Smartphone non-volatile storage.
- Confirms that the `Factory Data Reset` service performs a thorough file removal from the user data partition on Android Smartphones.

5.2 Future Work

This thesis provides a baseline into network forensics on Android devices. Future work to enhance this effort along with forensic examination on Android devices may expand this work in several ways:

- Development of acquisition methods that do not require rooting the device and are simple to use. The acquisition method used in this thesis did not alter the user data partition, but does require the device to be rooted which would could be legally challenged. Vidas Et al. explored the concept of flashing a custom boot image to forensically acquire data [39]. Their method would prove reliable in court, as it doesn't require rooting the device. However, custom boot images could be cumbersome for forensic experts to compile the required number of images for each different Android build and device.

- In an operational setting, a forensic examiner might be tasked to acquire data from the Android device before the suspect is apprehended. Acquisition methods would need to be employed without alerting the Smartphone owner. One such method would be to implant imaging software on the phone via the base station. This method would require access to the cellular base station and the ability to flash a custom boot image to the device. Recovering the data would pose another challenge to either push it back to the base station or save it to the SD card for later retrieval.

- A different clandestine acquisition method would require physical access to the device. A mobile computing platform could connect to the device via the USB interface and retrieve the contents of internal memory and return the device to the owner without his knowledge. This method would also have to flash a new boot image to the target device. Another challenge for this method would be the time required to retrieve data. A tool would need to be made that could ignore empty space in internal memory and pull all allocated and unallocated data from flash. One method that fits this description is acquiring data at charging kiosks [40].

- The development of a YAFFS2 forensic tool that provides a forensic examiner with retrieved data that directly relates to a file currently allocated on the file system or was previously allocated. It would be beneficial to modularize the tool to allow capture of personal information, network metadata or both as indicated by the operator. This tool will allow examiners to provide more thorough data analysis from recovered YAFFS2

partitions. Data analysis conducted in this thesis required more operator oriented analysis which would not be efficient when analyzing large amounts of data on multiple devices. One approach would be to develop a YAFFS2 implementation for the Sleuth Kit.

- Empirical tests have shown numerous unallocated Android location package files `cache.cell` and `cache.wifi` persist on flash memory. The experiments conducted in this thesis did not explore the use of multiple cellular base stations, and had only one instance of each cache file. If the Smartphones, with an associated Google account were allowed to operate for a substantial period of time, that data would prove very beneficial to forensic examiners. It would also be beneficial to understand the criteria required for the files to be written to memory.

- The experiments conducted in this thesis have shown when Bluetooth files are transferred form an external device, the Android OS performs multiple saves of the `btopp.db` file that stores the metadata from the transfer. Understanding why this database writes instances to memory while transferring data across bluetooth might prove to be beneficial.

- The data retrieved in this thesis might lead the very capable Android community to compile and use boot images to circumvent network metadata from writing network metadata to non-volatile memory. Methods such as these have become known as "anti-forensics" and could prove to be a detriment to forensic investigators. Methods to efficiently extract volatile data could provide a counter-action to anti-forensic techniques but would not provide the residual data found in this thesis. Another counter-action would involve flashing a customized boot image to a known criminals device as discussed previously.

With the growing use of the Android operating system at home and abroad, there is a pressing need for more advanced tools to analyze Android-based phones. It is hoped that the methodology presented in this thesis will help accomplish that goal.

REFERENCES

[1] IDC Estimates 50% Growth in Worldwide Smartphone Market in 2011, March 29, 2011. http://www.mobilemarketingwatch.com/idc-estimates-50-growth-in-worldwide-smartphone-market-in-2011-14227/.

[2] Wireless geographic logging engine, March 29, 2011. http://wigle.net/.

[3] Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, and E. Lear. Address allocation for private internets. RFC-1918, February 1996. http://www.ietf.org/rfc/rfc1918.txt.

[4] J. Wilcox. Gartner: Android smartphone sales surged 888.8% in 2010, April 1, 2011. http://www.betanews.com/joewilcox/article/Gartner-Android-smartphone-sales-surged-8888-in-2010/1297309933.

[5] W. Rothman. Smart phone malware: The six worst offenders, February 16, 2011. http://technolog.msnbc.msn.com/_news/2011/02/16/6063185-smart-phone-malware-the-six-worst-offenders.

[6] NIST. Computer Forensics Tool Testing, April 10, 2011. http://www.cftt.nist.gov/mobile_devices.htm.

[7] W. Jansen and R. Ayers. Guidelines on Cell Phone Forensics, May 10, 2007, NIST. Special Publication 800-101.

[8] A. Hoog. Android Forensics, 2011, Syngress.

[9] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti. Introduction to flash memory. *Proceedings of the IEEE*, 91(4):489 – 502, April 2003. ISSN 0018-9219.

[10] J.E. Regan. The forensic potential of flash memory. Master's thesis, Naval Postgraduate School, 2009.

[11] C. Manning. How YAFFS Works, March 15, 2011. http://www.yaffs.net/how-yaffs-works-internals.

[12] Android 2.3 Gingerbread to use EXT4 file system, May 22, 2011. http://www.h-online.com/open/news/item/Android-2-3-Gingerbread-to-use-Ext4-file-system-1152775.html.

[13] A. Mather, M. Cao, S. Bhattacharya, A Dilger, A Tomas, and L. Vivier. The new EXT4 filesystem: current status and future plans. 2:21–33, June 2007. ISSN 1530-2075. The Proceedings of the 2007 Linux Symposium.

[14] K. D. Fairbanks, C. P. Lee, and H. L. Owen, III. Forensic implications of EXT4. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, pp. 22:1–22:4. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0017-9. http://doi.acm.org/10.1145/1852666.1852691.

[15] J. Reardon, C. Marforio, S. Capkun, and D. Basin. Secure Deletion on Log-structured File Systems. *ArXiv e-prints*, June 2011.

[16] M. Al-Zarouni and H. Al-Hajri. Introduction to mobile phone flasher devices and considerations for their use in mobile phone forensics. 5th Australian Digital Forensics Conference, 2007.

[17] IEEE Standard Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2001*, pp. i –200, 2001.

[18] M. Breeuwsma, M. de Jongh, C. Klaver, R. van der Knijff, and M. Roeloffs. Forensic data recovery from flash memory. *Small Scale Digital Device Forensics Journal*, 1(1), 2007.

[19] Smart phone tool specification, April 11, 2010, NIST.

[20] G. Me and M. Rossi. Internal forensic acquisition for mobile equipments. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1 –7, April 2008. ISSN 1530-2075.

[21] A. Distefano and G. Me. An overall assessment of mobile internal acquisition tool. *Digital Investigation*, 5(Supplement 1):S121 – S127, 2008. ISSN 1742-2876. `http://www.sciencedirect.com/science/article/B7CW4-4T5SYCF-9/2/c8e1c13a779f5a9dda861da786f0d909`. The Proceedings of the Eighth Annual DFRWS Conference.

[22] Android debug bridge, March 1, 2011. `http://developer.android.com/guide/developing/tools/adb.html`.

[23] ROM Manager, May 1, 2011. `https://market.android.com/details?id=com.koushikdutta.rommanager&hl=en`.

[24] Combined GSM and Mobile IP Mobility Handling in UMTS IP CN, 2011. `http://www.3gpp.org/ftp/Specs/archive/23_series/23.923/`.

[25] Mobile Network Codes (MNC) for the International Identification Plan for Public Networks and Subscriptions, June 2010, International Telecommunication Union.

[26] C. Perkins. Mobility Support for IPv4. RFC-3344, August 2002. `http://www.ietf.org/rfc/rfc3344.txt`.

[27] C.E. Perkins. Mobile IP. *Communications Magazine, IEEE*, 35(5):84 –99, May 1997. ISSN 0163-6804.

[28] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, June 2007, IEEE.

[29] IEEE Standard for Information Technology Telecommunications and Information Exchange. *IEEE Std 802.15.12002*, pp. 0-1, June 2002.

[30] S. Garfinkel. Stream-based digital media forensics with bulk_extractor, 2011. In Submission.

[31] *bulk_extractor*, February 2, 2011. `http://afflib.org/software/bulk_extractor`.

[32] R. Beverly, S. Garfinkel, and G. Cardwell. Forensic carving of network packets and associated data structures. *Digital Investigation*, 8(Supplement 1):S78 – S89, 2011. ISSN 1742-2876. http://www.sciencedirect.com/science/article/pii/S174228761100034X. The Proceedings of the Eleventh Annual DFRWS Conference, 11th Annual Digital Forensics Research Conference.

[33] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 6(Supplement 1):S2 – S11, 2009. ISSN 1742-2876. http://www.sciencedirect.com/science/article/B7CW4-4X1HY5C-3/2/090ebc16025d598c775d87c8abbb7ae5. The Proceedings of the Ninth Annual DFRWS Conference.

[34] TacBSR Deployable Cellular Network (DCN), July 25, 2011. http://www.lgsinnovations.com/products-services/mobility/tacbsr-deployable-cellular-network-dcn.

[35] How To Root Nexus S With A Single Click Using Superboot, June 2, 2011. http://www.addictivetips.com/mobile/how-to-root-nexus-s-with-a-single-click-using-superboot/.

[36] Nakodari. How To: Motorola Atrix 4G Root, 2011. http://briefmobile.com/motorola-atrix-4g-root.

[37] jroid. [REF] [ROM] aosp 2.3.2 .img's for crespo (Nexus S), May 1, 2010. http://forum.xda-developers.com/showthread.php?t=884194.

[38] R. Droms. Dynamic host configuration protocol. RFC-2131, March 1997. http://www.ietf.org/rfc/rfc2131.txt.

[39] T. Vidas, C. Zhang, and N. Christin. Toward a general collection methodology for android devices. *Digital Investigation*, 8(Supplement 1):S14 – S24, 2011. ISSN 1742-2876. http://www.sciencedirect.com/science/article/pii/S1742287611000272. The Proceedings of the Eleventh Annual DFRWS Conference, 11th Annual Digital Forensics Research Conference.

[40] Smartphones and Tablets Targets for Getting 'Juiced', August 29, 2011. http://www.darkreading.com/blog/231600393/smartphones-and-tablets-targets-for-getting-juiced.html.

THIS PAGE INTENTIONALLY LEFT BLANK

Referenced Authors

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudly Knox Library
   Naval Postgraduate School
   Monterey, California